目錄

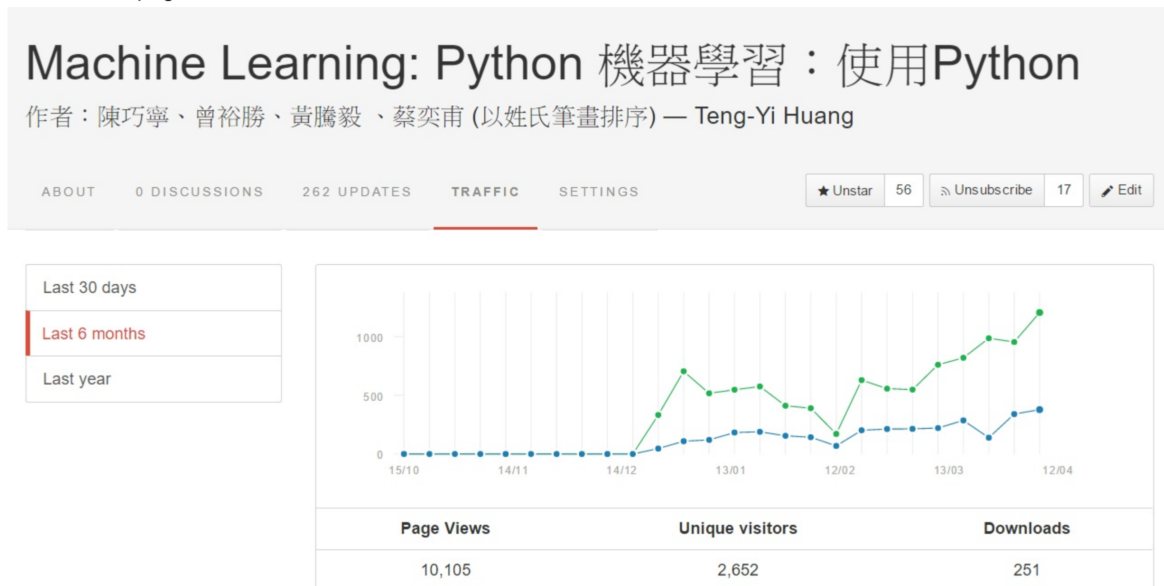
機器學習：使用Python

這份文件的目的是要提供Python 之機器學習套件 scikit-learn (http://scikit-learn.org/) 的中文使用說明。一開始的主要目標是詳細說明scikit-learn套件中的範例程式的使用流程以及相關函式的使用方法。目前使用版本為 scikit-learn version 0.19 以上

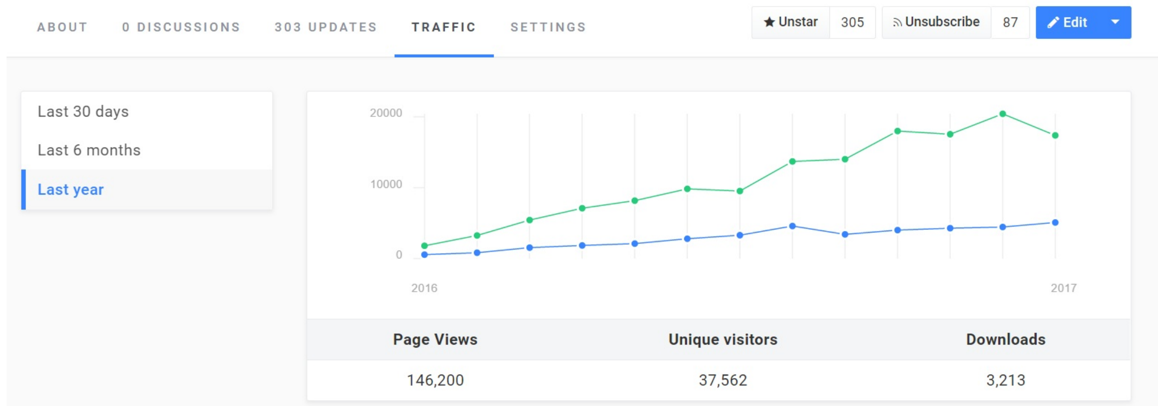
本書原始資料在 Github 上公開，歡迎大家共同參與維護： https://github.com/htygithub/machine-learning-python。

本文件主要的版本發展

- 0.0: 2015/12/21
  - 開始本文件「機器學習：使用Python」的撰寫
  - 初期以scikit-learn套件的範例介紹為主軸
- 0.1: 2016/4/15
  - 「機器學習：使用Python」文件
  - Contributor: 陳巧寧、曾裕勝、黃騰毅 、蔡奕甫
  - 新增章節: Classification, Clustering, cross_decomposition, Datasets, feature_selection, general_examples
  - 新增 introduction: 說明簡易的Anaconda安裝，以及利用數字辨識範例來入門機器學習的方法
  - 第 10,000個 pageview 達成



- 0.2: 2016/8/30
  - 新增應用章節，Contributor: 吳尚真
  - 增修章節: Classification, Datasets, feature_selection, general_examples
- 0.3: 2017/2/16
  - 新增應用章節，Contributor: 楊采玲、歐育年
  - 增修章節: Neural_Network, Decision tree
  - 2016年，使用者約四萬人次，頁面流量約15萬次。



Scikit-learn 套件

Scikit-learn (http://scikit-learn.org/) 是一個機器學習領域的開源套件。整個專案起始於 2007年由David Cournapeau所執行的 `Google Summer of Code` 計畫。而2010年之後，則由法國國家資訊暨自動化研究院（INRIA, http://www.inria.fr） 繼續主導及後續的支援及開發。近幾年(2013-2015)則由 INRIA 支持 Olivier Grisel (http://ogrisel.com) 全職負責該套件的維護工作。以開發者的角度來觀察，會發現Scikit-learn的整套使用邏輯設計的極其簡單。往往能將繁雜的機器學習理論簡化到一個步驟完成。Python的機器學習相關套件相當多，為何Scikit-learn會是首選之一呢？其實一個開源套件的選擇，最簡易的指標就是其 `contributor: 貢獻者` 、 `commits:版本數量` 以及最新的更新日期。下圖是2016/1/3 經過了美好的跨年夜後，筆者於官方開源程式碼網站(https://github.com/scikit-learn/scikit-learn) 所擷取的畫面。我們可以發現最新 `commit` 是四小時前，且 `contributor` 及 `commit` 數量分別為531人及 20,331個。由此可知，至少在2016年，這個專案乃然非常積極的在運作。在眾多機器學習套件中，不論是貢獻者及版本數量皆是最龐大的。也因此是本文件介紹機器學習的切入點。未來，我們希望能介紹更多的機器學習套件以及理論，也歡迎有志之士共同參與維護。

scikit-learn / **scikit-learn** Watch 1,022 Star 9,021 Fork 5,334

Code | Issues 598 | Pull requests 390 | Wiki | Pulse | Graphs

scikit-learn: machine learning in Python http://scikit-learn.org

20,331 commits | 15 branches | 67 releases | 531 contributors

Branch: master | New pull request | New file | Find file | HTTPS | https://github.com/ | Download ZIP

GaelVaroquaux Merge pull request #6108 from chezou/chezou-patch-1 … Latest commit e464689 4 hours ago

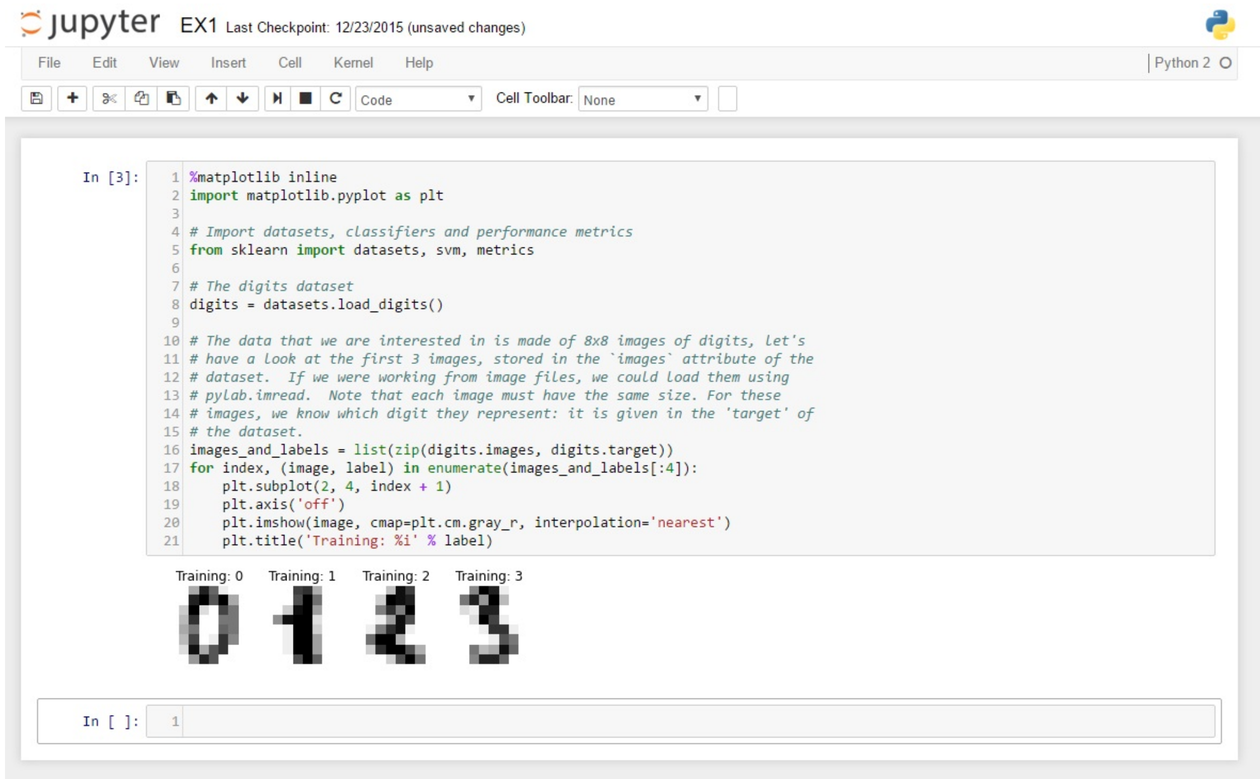
benchmarks	MAINT Use with to handle file	a month ago
continuous_integration	Merge pull request #5578 from waterponey/circleCI_push	23 days ago
doc	fixed target link to isotonic regression example	6 days ago
examples	FIX Eliminated the use of color_cycle in out of core example	6 days ago
sklearn	Remove extra `+` in document	18 hours ago
.coveragerc	coverall added	2 years ago
.gitattributes	ENH refactor NMF and add CD solver	3 months ago
.gitignore	add .tags in gitignore	a month ago
.landscape.yml	make landscape.io much more useful	10 months ago
.mailmap	Adding email to the mailmap	2 months ago
.travis.yml	Fix coveralls report sending	a month ago
AUTHORS.rst	Authors: Update based on #3067	2 years ago
CONTRIBUTING.md	Fix #4978: Typo in CONTRIBUTING.md	6 months ago

Scikit-learn 套件的安裝

目前Scikit-learn同時支援Python 2及 3，安裝的方式也非常多種。對於初學者，最建議的方式是直接下載 Anaconda Python (https://www.continuum.io/downloads)。同時支援 Windows / OSX/ Linux 等作業系統。相關數據分析套件如Scipy, Numpy, 及圖形繪製庫 matplotlib, bokeh 會同時安裝。

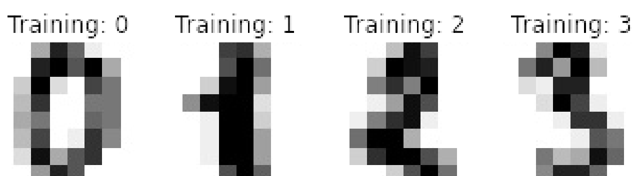
開發介面及環境

筆者目前最常用的開發介面為IPython Notebook (3.0版後已改名為Jupyter Notebook) 以及 Atom.io 文字編輯器。在安裝Anaconda啟用IPython Notebook介面後，本文件連結之程式碼皆能夠以複製貼上的方式執行測試。目前部份章節也附有notebook格式文件 `.ipynb` 檔可借下載。



給機器學習的初學者

本文件的目的並非探討機器學習的各項理論，我們將以應用範例著手來幫助學習。其中建議以手寫數字辨識來當成的敲門磚。而本文件中，有以下範例介紹手寫數字辨識，並且藉由這個應用來探討機器學習中的一個重要類別「監督式學習」。一開始，建議先從 機器學習資料集 Datasets，來了解資料集的型態以及取得方式。接下來最重要的是釐清特徵 `X` 以及預測目標 `y` 之間的關係。要注意這邊的大寫的 `X` 通常代表一個矩陣, 每一列代表一筆資料，而每一行則代表其特徵。例如手寫數字辨識是利用 8x8的影像資料，來當成訓練集。而其中一種特徵的取用方法是例用這64個像素的灰階值來當成特徵。而小寫的 `y` 則代表一個向量，這個向量紀錄著前述訓練資料對應的「答案」。



了解資料集之後，接下來則建議先嘗試 分類法範例一例用最簡單的支持向量機(Support Vector Machine)分類法來達成多目標分類 (Multi-class classification)，這裏的「多目標」指的是0到9的數字，該範例利用Scikit-learn內建的SVM分類器，來找出十個目標的分類公式，並介紹如何評估分類法的準確度，以及一些常見的分類指標。例如以下報表標示著對於10個數字的

預測準確度。 有了對這個範例的初步認識之後，讀者應該開始感覺到監督式學習(Supervised learning)的意義，這裏「監督」的意思是，我們已經知道資料所對應的預測目標，也就是利用圖形可猜出數字。也就是訓練集中有 `y` 。而另一大類別「非監督式學習」則是我們一開始並不知道 `y` ，我們想透過演算法來將 `y` 找出來。例如透過購買行爲及個人資料來分類消費族群。

```
             precision    recall  f1-score   support

          0       1.00      0.99      0.99        88
          1       0.99      0.97      0.98        91
          2       0.99      0.99      0.99        86
          3       0.98      0.87      0.92        91
          4       0.99      0.96      0.97        92
          5       0.95      0.97      0.96        91
          6       0.99      0.99      0.99        91
          7       0.96      0.99      0.97        89
          8       0.94      1.00      0.97        88
          9       0.93      0.98      0.95        92

avg / total       0.97      0.97      0.97       899
```

而有了基本的分類法，接下來的範例則是利用特徵選擇來更增進分類的準確性。以手寫數字辨識來說。上述的例子共使用了64個像素來當成特徵，然而以常理來判斷。這64個像素中，處於影像邊緣的像素參考價值應該不高，因爲手寫的筆畫鮮少出現在該處。若能將這些特徵資料排除在分類公式中，通常能再增進預測的準確度。而「特徵選擇」的這項技術，主要就是用來處理這類問題。特徵選擇範例二:Recursive Feature Elimination則是利用了Scikit-learn內建的特徵消去法，來找出消去那些特徵能夠最佳化預測的準確度。而 特徵選擇範例三：Recursive Feature Elimination with Cross-Validation 則使用了更進階的交叉驗證法來切分訓練集以及挑戰集來評估準確程度。建議讀者可以嘗試這幾個範例，一步步去深入機器學習的核心。

分類法 Classification

+

分類法/範例一: Recognizing hand-written digits

http://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html

這個範例用來展示scikit-learn 機器學習套件，如何用SVM演算法來達成手寫的數字辨識

1. 利用 `make_classification` 建立模擬資料
2. 利用 `sklearn.datasets.load_digits()` 來讀取內建資料庫
3. 用線性的SVC來做分類，以8x8的影像之像素值來當作特徵(共64個特徵)
4. 用 `metrics.classification_report` 來提供辨識報表

(一)引入函式庫及內建手寫數字資料庫

引入之函式庫如下

1. matplotlib.pyplot: 用來繪製影像
2. sklearn.datasets: 用來繪入內建之手寫數字資料庫
3. sklearn.svm: SVM 支持向量機之演算法物件
4. sklearn.metrics: 用來評估辨識準確度以及報表的顯示

```
import matplotlib.pyplot as plt
from sklearn import datasets, svm, metrics

# The digits dataset
digits = datasets.load_digits()
```

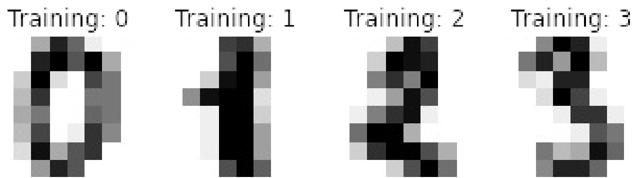
使用 `datasets.load_digits()` 將資料存入， `digits` 爲一個dict型別資料，我們可以用以下指令來看一下資料的內容。

```
for key,value in digits.items() :
    try:
        print (key,value.shape)
    except:
        print (key)
```

顯示	說明
('images', (1797L, 8L, 8L))	共有 1797 張影像，影像大小爲 8x8
('data', (1797L, 64L))	data 則是將8x8的矩陣攤平成64個元素之一維向量
('target_names', (10L,))	說明10種分類之對應 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
DESCR	資料之描述
('target', (1797L,))	記錄1797張影像各自代表那一個數字

接下來我們試著以下面指令來觀察資料檔，每張影像所對照的實際數字存在 `digits.target` 變數中

```
images_and_labels = list(zip(digits.images, digits.target))
for index, (image, label) in enumerate(images_and_labels[:4]):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % label)
```



(二)訓練以及分類

接下來的步驟則是使用 `reshape` 指令將8x8的影像資料攤平成64x1的矩陣。 接著用 `classifier = svm.SVC(gamma=0.001)` 產生一個SVC分類器(Support Vector Classification)。再將一半的資料送入分類器來訓練 `classifier.fit(資料:898x64, 分類目標:898x1)` 。SVC之預設kernel function爲RBF (radial basis function): $$\exp(-\gamma |x-x'|^2)$$. 其中 `SVC(gamma=0.001)` 就是在設定RBF函數裏的$$\gamma$$ 這個值必需要大於零。最後，再利用後半部份的資料來測試訓練完成之SVC分類機 `predict(data[n_samples / 2:])` 將預測結果存入 `predicted` 變數，而原先的真實目標資料則存於 `expected` 變數，用於下一節之準確度統計。

```
n_samples = len(digits.images)

# 資料攤平:1797 x 8 x 8 -> 1797 x 64
# 這裏的-1代表自動計算，相當於 (n_samples, 64)
data = digits.images.reshape((n_samples, -1))

# 產生SVC分類器
classifier = svm.SVC(gamma=0.001)

# 用前半部份的資料來訓練
classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])

expected = digits.target[n_samples // 2:]

#利用後半部份的資料來測試分類器，共 899筆資料
predicted = classifier.predict(data[n_samples // 2:])
```

若是觀察 `expected` 及 `predicted` 矩陣中之前10個變數可以得到:

- `expected[:10]` :[8 8 4 9 0 8 9 8 1 2]
- `predicted[:10]` :[8 8 4 9 0 8 9 8 1 2]

這說明了前10個元素中，我們之前訓練完成的分類機，正確的分類了手寫數字資料。那對於全部測試資料的準確度呢？要如何量測？

(三)分類準確度統計

那在判斷準確度方面，我們可以使用一個名爲「混淆矩陣」(Confusion matrix)的方式來統計。

```
print("Confusion matrix:\n%s"
    % metrics.confusion_matrix(expected, predicted))
```

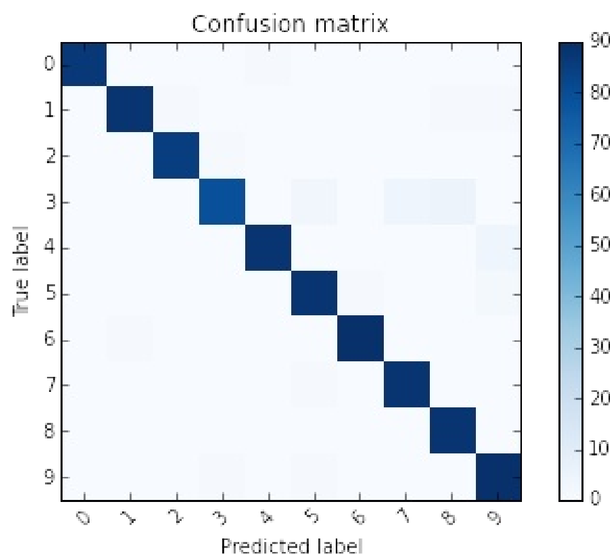
使用sklearn中之metrics物件， `metrics.confusion_matrix(真實資料:899, 預測資料:899)` 可以列出下面矩陣。此矩陣對角線左上方第一個數字 87，代表實際爲0且預測爲0的總數有87個，同一列(row)第五個元素則代表，實際爲0但判斷爲4的資料個數爲1個。

```
Confusion matrix:
[[87  0  0  0  1  0  0  0  0  0]
 [ 0 88  1  0  0  0  0  0  1  1]
 [ 0  0 85  1  0  0  0  0  0  0]
 [ 0  0  0 79  0  3  0  4  5  0]
 [ 0  0  0  0 88  0  0  0  0  4]
 [ 0  0  0  0  0 88  1  0  0  2]
 [ 0  1  0  0  0  0 90  0  0  0]
 [ 0  0  0  0  0  1  0 88  0  0]
 [ 0  0  0  0  0  0  0  0 88  0]
 [ 0  0  0  1  0  1  0  0  0 90]]
```

我們可以利用以下的程式碼將混淆矩陣圖示出來。由圖示可以看出，實際爲3時，有數次誤判爲5,7,8。

```
def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    import numpy as np
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(digits.target_names))
    plt.xticks(tick_marks, digits.target_names, rotation=45)
    plt.yticks(tick_marks, digits.target_names)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

plt.figure()
plot_confusion_matrix(metrics.confusion_matrix(expected, predicted))
```



以手寫影像3爲例，我們可以用四個數字來探討判斷的精準度。

1. True Positive(TP,眞陽):實際爲3且判斷爲3，共79個
2. False Positive(FP,僞陽):判斷爲3但判斷錯誤，共2個
3. False Negative(FN,僞陰):實際爲3但判斷錯誤，共12個
4. True Negative(TN,眞陰):實際不爲3，且判斷正確。也就是其餘899-79-2-12=885個

而在機器學習理論中，我們通常用以下precision, recall, f1-score來探討精確度。以手寫影像3爲例。

- precision = TP/(TP+FP) = 79/81 = 0.98
- 判斷爲3且實際爲3的比例爲0.98
- recall = TP/(TP+FN) = 79/91 = 0.87
- 實際爲3且判斷爲3的比例爲0.87
- f1-score 則爲以上兩者之「harmonic mean 調和平均數」
- f1-score= 2 x precision x recall/(recision + recall) = 0.92

metrics物件裏也提供了方便的函式 `metrics.classification_report(expected, predicted)` 計算以上統計數據。

```
print("Classification report for classifier %s:\n%s\n"
    % (classifier, metrics.classification_report(expected, predicted)))
```

此報表最後的 support，則代表著實際爲手寫數字的總數。例如實際爲3的數字共有91個。

```
             precision    recall  f1-score   support

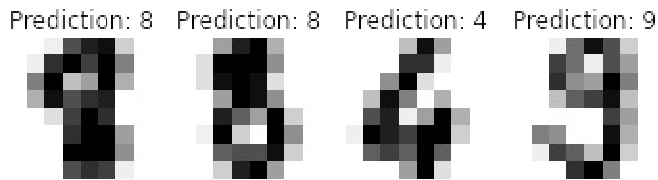
          0       1.00      0.99      0.99        88
          1       0.99      0.97      0.98        91
          2       0.99      0.99      0.99        86
          3       0.98      0.87      0.92        91
          4       0.99      0.96      0.97        92
          5       0.95      0.97      0.96        91
          6       0.99      0.99      0.99        91
          7       0.96      0.99      0.97        89
          8       0.94      1.00      0.97        88
          9       0.93      0.98      0.95        92

avg / total       0.97      0.97      0.97       899
```

最後，用以下的程式碼可以觀察測試影像以及預測(分類)結果得對應關係。

```
images_and_predictions = list(
                        zip(digits.images[n_samples // 2:], predicted))
for index, (image, prediction) in enumerate(images_and_predictions[:4]):
    plt.subplot(2, 4, index + 5)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Prediction: %i' % prediction)

plt.show()
```



(四)完整程式碼

Python source code: plot_digits_classification.py

http://scikit-learn.org/stable/_downloads/plot_digits_classification.py

```
print(__doc__)

# Author: Gael Varoquaux <gael dot varoquaux at normalesup dot org>
# License: BSD 3 clause

# Standard scientific Python imports
import matplotlib.pyplot as plt

# Import datasets, classifiers and performance metrics
from sklearn import datasets, svm, metrics

# The digits dataset
digits = datasets.load_digits()

# The data that we are interested in is made of 8x8 images of digits, let's
# have a look at the first 4 images, stored in the `images` attribute of the
# dataset.  If we were working from image files, we could load them using
# matplotlib.pyplot.imread.  Note that each image must have the same size. For these
# images, we know which digit they represent: it is given in the 'target' of
# the dataset.
images_and_labels = list(zip(digits.images, digits.target))
for index, (image, label) in enumerate(images_and_labels[:4]):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % label)

# To apply a classifier on this data, we need to flatten the image, to
# turn the data in a (samples, feature) matrix:
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Create a classifier: a support vector classifier
classifier = svm.SVC(gamma=0.001)

# We learn the digits on the first half of the digits
classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])

# Now predict the value of the digit on the second half:
expected = digits.target[n_samples // 2:]
```

```
predicted = classifier.predict(data[n_samples // 2:])

print("Classification report for classifier %s:\n%s\n"
      % (classifier, metrics.classification_report(expected, predicted)))
print("Confusion matrix:\n%s" % metrics.confusion_matrix(expected, predicted))

images_and_predictions = list(zip(digits.images[n_samples // 2:], predicted))
for index, (image, prediction) in enumerate(images_and_predictions[:4]):
    plt.subplot(2, 4, index + 5)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Prediction: %i' % prediction)

plt.show()
```

分類法/範例二: Normal and Shrinkage Linear Discriminant Analysis for classification

http://scikit-learn.org/stable/auto_examples/classification/plot_lda.html

這個範例用來展示scikit-learn 如何使用Linear Discriminant Analysis (LDA) 線性判別分析來達成資料分類的目的

1. 利用 `sklearn.datasets.make_blobs` 產生測試資料
2. 利用自定義函數 `generate_data` 產生具有數個特徵之資料集，其中僅有一個特徵對於資料分料判斷有意義
3. 使用 `LinearDiscriminantAnalysis` 來達成資料判別
4. 比較於LDA演算法中，開啓 `shrinkage` 前後之差異

(一)產生測試資料

從程式碼來看，一開始主要爲自定義函數 `generate_data(n_samples, n_features)` ，這個函數的主要目的爲產生一組測試資料，總資料列數爲 `n_samples` ，每一列共有 `n_features` 個特徵。而其中只有第一個特徵得以用來判定資料類別，其他特徵則毫無意義。 `make_blobs` 負責產生單一特徵之資料後，利用 ‘np.random.randn ‘ 亂數產生其他 ‘n_features - 1 ‘個特徵，之後利用 `np.hstack` 以"水平" (horizontal)方式連接X以及亂數產生之特徵資料。

```
%matplotlib inline
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

n_train = 20  # samples for training
n_test = 200  # samples for testing
n_averages = 50  # how often to repeat classification
n_features_max = 75  # maximum number of features
step = 4  # step size for the calculation

def generate_data(n_samples, n_features):
    X, y = make_blobs(n_samples=n_samples, n_features=1, centers=[[-2], [2]])
    # add non-discriminative features
    if n_features > 1:
        X = np.hstack([X, np.random.randn(n_samples, n_features - 1)])
    return X, y
```

我們可以用以下的程式碼來測試自定義函式，結果回傳了X (10x5矩陣)及y(10個元素之向量)，我們可以使用 `pandas.DataFrame` 套件來觀察資料

```
X, y = generate_data(10, 5)

import pandas as pd
pd.set_option('precision',2)
df=pd.DataFrame(np.hstack([y.reshape(10,1),X]))
df.columns = ['y', 'X0', 'X1', 'X2', 'X2', 'X4']
print(df)
```

結果顯示如下。。我們可以看到只有X的第一行特徵資料(X0) 與目標數值 y 有一個明確的對應關係，也就是y爲1時，數值較大。

```
   y    X0    X1    X2    X2    X4
0  1  0.38  0.35  0.80 -0.97 -0.68
1  1  2.41  0.31 -1.47  0.10 -1.39
2  1  1.65 -0.99 -0.12 -0.38  0.18
3  0 -4.86  0.14 -0.80  1.13 -1.31
4  1 -0.06 -1.99 -0.70 -1.26 -1.64
5  0 -1.51 -1.74 -0.83  0.74 -2.07
6  0 -2.50  0.44 -0.45 -0.55 -0.42
7  1  1.55  1.38  0.93 -1.44  0.27
8  0 -1.95  0.32 -0.28  0.02  0.07
9  0 -0.58 -0.07 -1.01  0.15 -1.84
```

(二)改變特徵數量並測試shrinkage之功能

接下來程式碼裏有兩段迴圈，外圈改變特徵數量。內圈則多次嘗試LDA之以求精準度。使用 `LinearDiscriminantAnalysis` 來訓練分類器，過程中以 `shrinkage='auto'` 以及 `shrinkage=None` 來控制shrinkage之開關，將分類器分別以 `clf1` 以及 `clf2` 儲存。之後再產生新的測試資料將準確度加入 `score_clf1` 及 `score_clf2` 裏，離開內迴圈之後除以總數以求平均。

```
acc_clf1, acc_clf2 = [], []
n_features_range = range(1, n_features_max + 1, step)
for n_features in n_features_range:
    score_clf1, score_clf2 = 0, 0
    for _ in range(n_averages):
        X, y = generate_data(n_train, n_features)

        clf1 = LinearDiscriminantAnalysis(solver='lsqr', shrinkage='auto').fit(X, y)
        clf2 = LinearDiscriminantAnalysis(solver='lsqr', shrinkage=None).fit(X, y)

        X, y = generate_data(n_test, n_features)
        score_clf1 += clf1.score(X, y)
        score_clf2 += clf2.score(X, y)

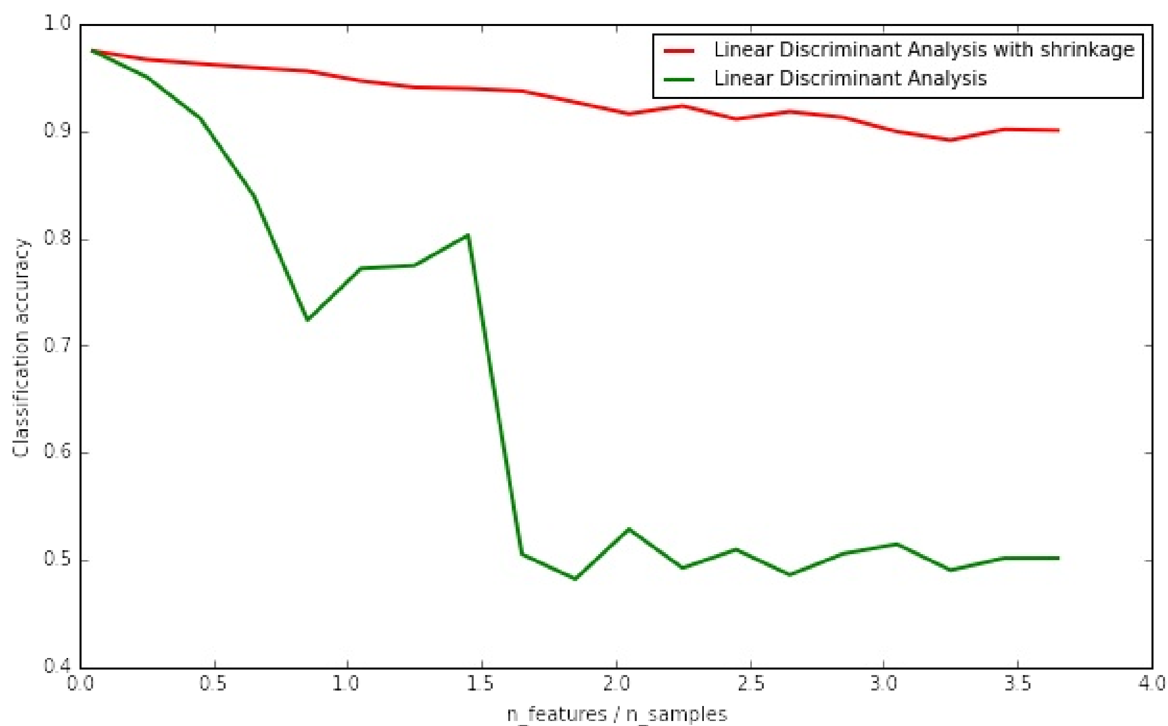
    acc_clf1.append(score_clf1 / n_averages)
    acc_clf2.append(score_clf2 / n_averages)
```

(三)顯示LDA判別結果

這個範例主要希望能得知 `shrinkage` 的功能，因此畫出兩條分類準確度的曲線。縱軸代表平均的分類準確度，而橫軸代表的是 `features_samples_ratio` 顧名思義，它是模擬資料中，特徵數量與訓練資料列數的比例。當特徵數量爲75且訓練資料列數僅有20筆時， `features_samples_ratio = 3.75` 由於資料列數過少，導致準確率下降。而此時 `shrinkage` 演算法能有效維持LDA演算法的準確度。

```
features_samples_ratio = np.array(n_features_range) / n_train
fig = plt.figure(figsize=(10,6), dpi=300)
plt.plot(features_samples_ratio, acc_clf1, linewidth=2,
         label="Linear Discriminant Analysis with shrinkage", color='r')
plt.plot(features_samples_ratio, acc_clf2, linewidth=2,
         label="Linear Discriminant Analysis", color='g')
plt.xlabel('n_features / n_samples')
plt.ylabel('Classification accuracy')

plt.legend(loc=1, prop={'size': 10})
plt.show()
```



(四)完整程式碼

Python source code: plot_lda.py

```
from __future__ import division

import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis


n_train = 20  # samples for training
n_test = 200  # samples for testing
n_averages = 50  # how often to repeat classification
n_features_max = 75  # maximum number of features
step = 4  # step size for the calculation
```

```python
def generate_data(n_samples, n_features):
    """Generate random blob-ish data with noisy features.

    This returns an array of input data with shape `(n_samples, n_features)`
    and an array of `n_samples` target labels.

    Only one feature contains discriminative information, the other features
    contain only noise.
    """
    X, y = make_blobs(n_samples=n_samples, n_features=1, centers=[[-2], [2]])

    # add non-discriminative features
    if n_features > 1:
        X = np.hstack([X, np.random.randn(n_samples, n_features - 1)])
    return X, y

acc_clf1, acc_clf2 = [], []
n_features_range = range(1, n_features_max + 1, step)
for n_features in n_features_range:
    score_clf1, score_clf2 = 0, 0
    for _ in range(n_averages):
        X, y = generate_data(n_train, n_features)

        clf1 = LinearDiscriminantAnalysis(solver='lsqr', shrinkage='auto').fit(X, y)
        clf2 = LinearDiscriminantAnalysis(solver='lsqr', shrinkage=None).fit(X, y)

        X, y = generate_data(n_test, n_features)
        score_clf1 += clf1.score(X, y)
        score_clf2 += clf2.score(X, y)

    acc_clf1.append(score_clf1 / n_averages)
    acc_clf2.append(score_clf2 / n_averages)

features_samples_ratio = np.array(n_features_range) / n_train

plt.plot(features_samples_ratio, acc_clf1, linewidth=2,
         label="Linear Discriminant Analysis with shrinkage", color='r')
plt.plot(features_samples_ratio, acc_clf2, linewidth=2,
         label="Linear Discriminant Analysis", color='g')

plt.xlabel('n_features / n_samples')
plt.ylabel('Classification accuracy')

plt.legend(loc=1, prop={'size': 12})
plt.suptitle('Linear Discriminant Analysis vs. \
shrinkage Linear Discriminant Analysis (1 discriminative feature)')
plt.show()
```

分類法/範例三: Plot classification probability

這個範例的主要目的

- 使用iris 鳶尾花資料集
- 測試不同分類器對於涵蓋特定範圍之資料集，分類爲那一種鳶尾花的機率
- 例如：sepal length 爲 4cm 而 sepal width 爲 3cm時被分類爲 versicolor的機率

(一)資料匯入及描述

- 首先先匯入iris 鳶尾花資料集，使用 `iris = datasets.load_iris()` 將資料存入
- 準備X (特徵資料) 以及 y (目標資料)，僅使用兩個特徵方便視覺呈現

```
import matplotlib.pyplot as plt
import numpy as np

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data[:, 0:2]  # 僅使用前兩個特徵，方便視覺化呈現
y = iris.target

n_features = X.shape[1]
```

- `iris` 爲一個dict型別資料，我們可以用以下指令來看一下資料的內容。

```
for key,value in iris.items() :
    try:
        print (key,value.shape)
    except:
        print (key)
```

顯示	說明
('target_names', (3L,))	共有三種鳶尾花 setosa, versicolor, virginica
('data', (150L, 4L))	有150筆資料，共四種特徵
('target', (150L,))	這150筆資料各是那一種鳶尾花
DESCR	資料之描述
feature_names	四個特徵代表的意義

(二) 分類器的選擇

這個範例選擇了四種分類器，存入一個dict資料中，分別爲：

1. L1 logistic
2. L2 logistic (OvR)
3. Linear SVC

4. L2 logistic (Multinomial)

其中 `LogisticRegression` 並不適合拿來做多目標的分類器，我們可以用結果圖的分類機率來觀察。

```
C = 1.0

# Create different classifiers. The logistic regression cannot do
# multiclass out of the box.
classifiers = {'L1 logistic': LogisticRegression(C=C, penalty='l1'),
               'L2 logistic (OvR)': LogisticRegression(C=C, penalty='l2'),
               'Linear SVC': SVC(kernel='linear', C=C, probability=True,
                                 random_state=0),
               'L2 logistic (Multinomial)': LogisticRegression(
                C=C, solver='lbfgs', multi_class='multinomial'
                )}

n_classifiers = len(classifiers)
```

而接下來爲了產生一個包含絕大部份可能的測試矩陣，我們會用到以下指令。

1. `np.linspace(起始, 終止, 數量)` 目的爲產生等間隔之數據，例如 `print(np.linspace(1,3,3))` 的結果爲 `[ 1. 2. 3.]`，而 `print(np.linspace(1,3,5))` 的結果爲 `[ 1. 1.5 2. 2.5 3. ]`
2. `np.meshgrid(xx,yy)` 則用來產生網格狀座標。
3. `numpy.c_` 爲numpy特殊物件，能協助將numpy 陣列連接起來，將程式簡化後，我們用以下範例展示相關函式用法。

```
xx, yy = np.meshgrid(np.linspace(1,3,3), np.linspace(4,6,3).T)
Xfull = np.c_[xx.ravel(), yy.ravel()]
print('xx= \n%s\n' % xx)
print('yy= \n%s\n' % yy)
print('xx.ravel()= %s\n' % xx.ravel())
print('Xfull= \n%s' % Xfull)
```

結果顯示如下，我們可以看出Xfull模擬出了一個類似特徵矩陣X, 具備有9筆資料，這九筆資料重現了xx (3種數值變化)及yy(3種數值變化)的所有排列組合。

```
xx=
[[ 1.  2.  3.]
 [ 1.  2.  3.]
 [ 1.  2.  3.]]

yy=
[[ 4.  4.  4.]
 [ 5.  5.  5.]
 [ 6.  6.  6.]]

xx.ravel()= [ 1.  2.  3.  1.  2.  3.  1.  2.  3.]

Xfull=
[[ 1.  4.]
 [ 2.  4.]
 [ 3.  4.]
 [ 1.  5.]
 [ 2.  5.]
 [ 3.  5.]
 [ 1.  6.]
 [ 2.  6.]
 [ 3.  6.]]
```

而下面這段程式碼的主要用意，在產生一個網格矩陣，其中xx,yy分別代表著 `iris` 資料集的第一及第二個特徵。xx 是3~9之間的100個連續數字，而yy是1~5之間的100個連續數字。用 `np.meshgrid(xx,yy)` 及 `np.c_` 產生出Xfull特徵矩陣，10,000筆資料包含了兩個特徵的所有排列組合。

```
plt.figure(figsize=(3 * 2, n_classifiers * 2))
plt.subplots_adjust(bottom=.2, top=.95)

xx = np.linspace(3, 9, 100)
yy = np.linspace(1, 5, 100).T
xx, yy = np.meshgrid(xx, yy)
Xfull = np.c_[xx.ravel(), yy.ravel()]
```

(三) 測試分類器以及畫出機率分佈圖的選擇

接下來的動作

1. 用迴圈輪過所有的分類器，並計算顯示分類成功率
2. 將 `Xfull` (10000x2矩陣)傳入 `classifier.predict_proba()` 得到 `probas` (10000x3矩陣)。這裏的 `probas` 矩陣是10000種不同的特徵排列組合所形成的數據，被分類到三種iris 鳶尾花的可能性。
3. 利用 `reshape((100,100))` 將10000筆資料排列成二維矩陣，並將機率用影像的方式呈現出來

```
#若在ipython notebook (Jupyter) 裏執行，則可以將下列這行的井號移除
%matplotlib inline
#原範例沒有下列這行，這是爲了讓圖形顯示更漂亮而新增的
fig = plt.figure(figsize=(12,12), dpi=300)

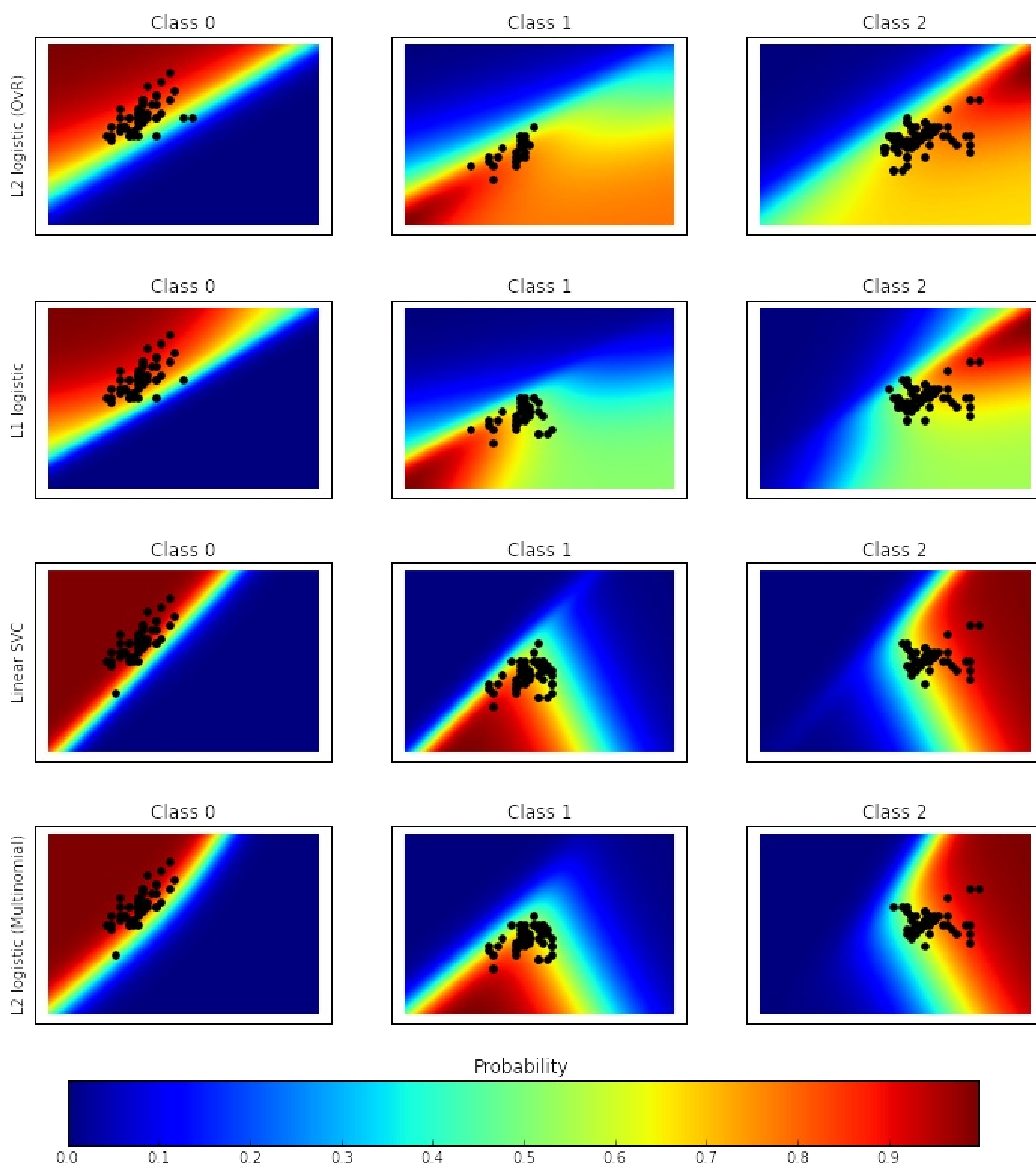
for index, (name, classifier) in enumerate(classifiers.items()):
    #訓練並計算分類成功率
    #然而此範例訓練跟測試用相同資料集，並不符合實際狀況。
    #建議採用cross_validation的方式才能較正確評估
    classifier.fit(X, y)
    y_pred = classifier.predict(X)
    classif_rate = np.mean(y_pred.ravel() == y.ravel()) * 100
    print("classif_rate for %s : %f " % (name, classif_rate))

    # View probabilities=
    probas = classifier.predict_proba(Xfull)
    n_classes = np.unique(y_pred).size
    for k in range(n_classes):
        plt.subplot(n_classifiers, n_classes, index * n_classes + k + 1)
        plt.title("Class %d" % k)
        if k == 0:
            plt.ylabel(name)
        imshow_handle = plt.imshow(probas[:, k].reshape((100, 100)),
                                   extent=(3, 9, 1, 5), origin='lower')
        plt.xticks(())
        plt.yticks(())
        idx = (y_pred == k)
        if idx.any():
            plt.scatter(X[idx, 0], X[idx, 1], marker='o', c='k')

ax = plt.axes([0.15, 0.04, 0.7, 0.05])
plt.title("Probability")
plt.colorbar(imshow_handle, cax=ax, orientation='horizontal')

plt.show()
```

```
classif_rate for L2 logistic (OvR) : 76.666667
classif_rate for L1 logistic : 79.333333
classif_rate for Linear SVC : 82.000000
classif_rate for L2 logistic (Multinomial) : 82.000000
```



(四)完整程式碼

Python source code: plot_classification_probability.py

http://scikit-learn.org/stable/_downloads/plot_classification_probability.py

```
print(__doc__)

# Author: Alexandre Gramfort <alexandre.gramfort@inria.fr>
# License: BSD 3 clause

import matplotlib.pyplot as plt
import numpy as np
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data[:, 0:2]  # we only take the first two features for visualization
y = iris.target

n_features = X.shape[1]

C = 1.0

# Create different classifiers. The logistic regression cannot do
# multiclass out of the box.
classifiers = {'L1 logistic': LogisticRegression(C=C, penalty='l1'),
               'L2 logistic (OvR)': LogisticRegression(C=C, penalty='l2'),
               'Linear SVC': SVC(kernel='linear', C=C, probability=True,
                                 random_state=0),
               'L2 logistic (Multinomial)': LogisticRegression(
                C=C, solver='lbfgs', multi_class='multinomial'
                )}

n_classifiers = len(classifiers)

plt.figure(figsize=(3 * 2, n_classifiers * 2))
plt.subplots_adjust(bottom=.2, top=.95)

xx = np.linspace(3, 9, 100)
yy = np.linspace(1, 5, 100).T
xx, yy = np.meshgrid(xx, yy)
Xfull = np.c_[xx.ravel(), yy.ravel()]

for index, (name, classifier) in enumerate(classifiers.items()):
    classifier.fit(X, y)

    y_pred = classifier.predict(X)
    classif_rate = np.mean(y_pred.ravel() == y.ravel()) * 100
    print("classif_rate for %s : %f " % (name, classif_rate))

    # View probabilities=
    probas = classifier.predict_proba(Xfull)
    n_classes = np.unique(y_pred).size
    for k in range(n_classes):
        plt.subplot(n_classifiers, n_classes, index * n_classes + k + 1)
        plt.title("Class %d" % k)
        if k == 0:
            plt.ylabel(name)
        imshow_handle = plt.imshow(probas[:, k].reshape((100, 100)),
                                   extent=(3, 9, 1, 5), origin='lower')
        plt.xticks(())
        plt.yticks(())
        idx = (y_pred == k)
        if idx.any():
```

```
            plt.scatter(X[idx, 0], X[idx, 1], marker='o', c='k')

ax = plt.axes([0.15, 0.04, 0.7, 0.05])
plt.title("Probability")
plt.colorbar(imshow_handle, cax=ax, orientation='horizontal')

plt.show()
```

分類法/範例四: Classifier comparison

這個範例的主要目的

- 比較各種分類器
- 利用圖示法觀察各種分類器的分類邊界及區域

(一)引入函式並準備分類器

- 將分類器引入之後存放入一個 `list` 裏
- 這邊要注意 sklearn.discriminant_analysis 必需要 `sklearn 0.17` 以上才能執行

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

h = .02  # step size in the mesh

names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Decision Tree",
         "Random Forest", "AdaBoost", "Naive Bayes", "Linear Discriminant Ana.",
         "Quadratic Discriminant Ana."]
classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
    AdaBoostClassifier(),
    GaussianNB(),
    LinearDiscriminantAnalysis(),
    QuadraticDiscriminantAnalysis()]
```

(二)準備測試資料

- 利用 `make_classification` 產生分類資料， `n_features=2` 表示共有兩個特徵, `n_informative=2` 代表有兩個類別
- 所產生之 X: 100 x 2矩陣，y: 100 元素之向量，y的數值僅有0或是1用來代表兩種類別
- 利用 `X += 2 * rng.uniform(size=X.shape)` 加入適度的雜訊後將(X,y)資料集命名爲 `linear_separable`
- 最後利用 `make_moon()` 及 `make_circles()` 產生空間中月亮形狀及圓形之數據分佈後，一併存入 `datasets` 變數

```
X, y = make_classification(n_features=2, n_redundant=0, n_informative=2,
                           random_state=1, n_clusters_per_class=1)
rng = np.random.RandomState(2)
X += 2 * rng.uniform(size=X.shape)
linearly_separable = (X, y)

datasets = [make_moons(noise=0.3, random_state=0),
            make_circles(noise=0.2, factor=0.5, random_state=1),
            linearly_separable
            ]
```

(三)測試分類器並作圖

接下來這段程式碼有兩個for 迴圈，外迴圈走過三個的dataset，內迴圈則走過所有的分類器。 爲求簡要說明，我們將程式碼簡略如下：

1. 外迴圈：資料迴圈。首先畫出資料分佈，接著將資料傳入分類器迴圈

```
for ds in datasets:
    X, y = ds
    #調整特徵值大小使其在特定範圍
    X = StandardScaler().fit_transform(X)
    #利用train_test_split將資料分成訓練集以及測試集
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4)
    #產生資料網格來大範圍測試分類器，範例EX 3有詳述該用法
    xx, yy = np.meshgrid(.........省略)
    # 畫出訓練資料點
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
    # 畫出測試資料點，用alpha=0.6將測試資料點畫的"淡"一些
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6)
```

2. 內迴圈：分類器迴圈。測試分類準確度並繪製分類邊界及區域

```
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, m_max]x[y_min, y_max].
    if hasattr(clf, "decision_function"):
        Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
    else:
        Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)
```

爲了顯示方便，我將原始碼的內圈改爲 `for name, clf in zip(names[0:4], classifiers[0:4]):` 只跑過前四個分類器。

```
%matplotlib inline
```

```
figure = plt.figure(figsize=(30,20), dpi=300)
i = 1
# iterate over datasets
for ds in datasets:
    # preprocess dataset, split into training and test part
    X, y = ds
    X = StandardScaler().fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4)

    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    # just plot the dataset first
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])
    ax = plt.subplot(len(datasets), (len(classifiers) + 1)//2, i)
    # Plot the training points
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
    # and testing points
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6)
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    i += 1

    # iterate over classifiers
    for name, clf in zip(names[0:4], classifiers[0:4]):
        ax = plt.subplot(len(datasets), (len(classifiers) + 1)//2, i)
        clf.fit(X_train, y_train)
        score = clf.score(X_test, y_test)

        # Plot the decision boundary. For that, we will assign a color to each
        # point in the mesh [x_min, m_max]x[y_min, y_max].
        if hasattr(clf, "decision_function"):
            Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
        else:
            Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]

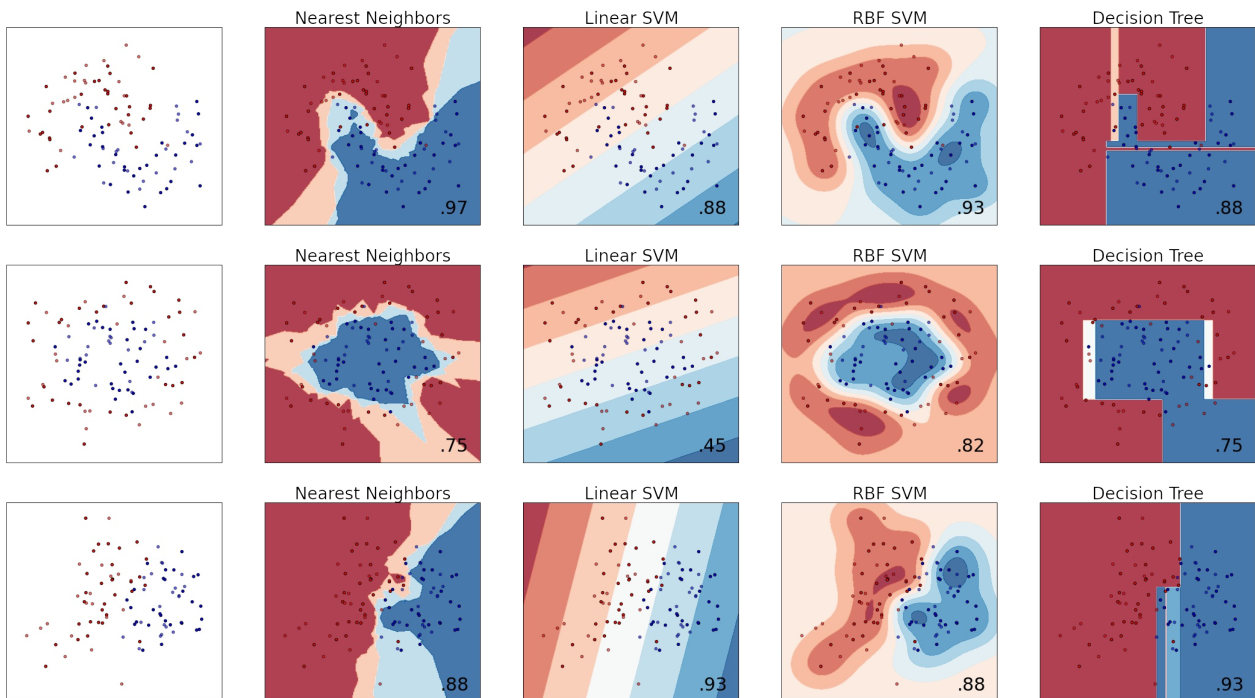
        # Put the result into a color plot
        Z = Z.reshape(xx.shape)
        ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)

        # Plot also the training points
        ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
        # and testing points
        ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright,
                   alpha=0.6)

        ax.set_xlim(xx.min(), xx.max())
```

```
        ax.set_ylim(yy.min(), yy.max())
        ax.set_xticks(())
        ax.set_yticks(())
        ax.set_title(name,fontsize=28)
        ax.text(xx.max() - .3, yy.min() + .3, ('%.2f' % score).lstrip('0'),
                size=30, horizontalalignment='right')
        i += 1

figure.subplots_adjust(left=.02, right=.98)
plt.show()
```



(四) 原始碼列表

Python source code: plot_classifier_comparison.py

http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

```
print(__doc__)


# Code source: Gaël Varoquaux
#              Andreas Müller
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

h = .02  # step size in the mesh

names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Decision Tree",
         "Random Forest", "AdaBoost", "Naive Bayes", "Linear Discriminant Analysis",
         "Quadratic Discriminant Analysis"]
classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
    AdaBoostClassifier(),
    GaussianNB(),
    LinearDiscriminantAnalysis(),
    QuadraticDiscriminantAnalysis()]

X, y = make_classification(n_features=2, n_redundant=0, n_informative=2,
                           random_state=1, n_clusters_per_class=1)
rng = np.random.RandomState(2)
X += 2 * rng.uniform(size=X.shape)
linearly_separable = (X, y)

datasets = [make_moons(noise=0.3, random_state=0),
            make_circles(noise=0.2, factor=0.5, random_state=1),
            linearly_separable
            ]

figure = plt.figure(figsize=(27, 9))
i = 1
# iterate over datasets
for ds in datasets:
    # preprocess dataset, split into training and test part
    X, y = ds
    X = StandardScaler().fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4)

    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    # just plot the dataset first
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])
    ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
    # Plot the training points
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
```

```
    # and testing points
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6)
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    i += 1

    # iterate over classifiers
    for name, clf in zip(names, classifiers):
        ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
        clf.fit(X_train, y_train)
        score = clf.score(X_test, y_test)

        # Plot the decision boundary. For that, we will assign a color to each
        # point in the mesh [x_min, m_max]x[y_min, y_max].
        if hasattr(clf, "decision_function"):
            Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
        else:
            Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]

        # Put the result into a color plot
        Z = Z.reshape(xx.shape)
        ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)

        # Plot also the training points
        ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
        # and testing points
        ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright,
                   alpha=0.6)

        ax.set_xlim(xx.min(), xx.max())
        ax.set_ylim(yy.min(), yy.max())
        ax.set_xticks(())
        ax.set_yticks(())
        ax.set_title(name)
        ax.text(xx.max() - .3, yy.min() + .3, ('%.2f' % score).lstrip('0'),
                size=15, horizontalalignment='right')
        i += 1

figure.subplots_adjust(left=.02, right=.98)
plt.show()
```

分類法/範例五：Linear and Quadratic Discriminant Analysis with confidence ellipsoid

線性判別以及二次判別的比較

http://scikit-learn.org/stable/auto_examples/classification/plot_lda_qda.html

(一)資料產生function

這個範例引入的套件，主要特點在：

1. `scipy.linalg` :線性代數相關函式，這裏主要使用到linalg.eigh 特徵值相關問題
2. `matplotlib.colors` : 用來處理繪圖時的色彩分佈
3. `LinearDiscriminantAnalysis` :線性判別演算法
4. `QuadraticDiscriminantAnalysis` :二次判別演算法

```
%matplotlib inline
from scipy import linalg
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import colors
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
```

接下來是設定一個線性變化的colormap， `LinearSegmentedColormap(name, segmentdata)` 預設會傳回一個256個值的數值顏色對應關係。用一個具備有三個項目的dict變數 `segmentdata` 來設定。以 `'red': [(0, 1, 1), (1, 0.7, 0.7)]` 來解釋，就是我們希望數值由0到1的過程，紅色通道將由1線性變化至0.7。

```
cmap = colors.LinearSegmentedColormap(
    'red_blue_classes',
    {'red': [(0, 1, 1), (1, 0.7, 0.7)],
     'green': [(0, 0.7, 0.7), (1, 0.7, 0.7)],
     'blue': [(0, 0.7, 0.7), (1, 1, 1)]})
plt.cm.register_cmap(cmap=cmap)
```

我們可以用以下程式碼來觀察。當輸入數值為 `np.arange(0,1.1,0.1)` 也就是0,0.1...,1.0 時RGB數值的變化情形。

```
values = np.arange(0,1.1,0.1)
cmap_values = mpl.cm.get_cmap('red_blue_classes')(values)
import pandas as pd
pd.set_option('precision',2)
df=pd.DataFrame(np.hstack((values.reshape(11,1),cmap_values)))
df.columns = ['Value', 'R', 'G', 'B', 'Alpha']
print(df)
```

```
    Value    R    G    B  Alpha
0     0.0  1.0  0.7  0.7      1
1     0.1  1.0  0.7  0.7      1
2     0.2  0.9  0.7  0.8      1
3     0.3  0.9  0.7  0.8      1
4     0.4  0.9  0.7  0.8      1
5     0.5  0.8  0.7  0.9      1
6     0.6  0.8  0.7  0.9      1
7     0.7  0.8  0.7  0.9      1
8     0.8  0.8  0.7  0.9      1
9     0.9  0.7  0.7  1.0      1
10    1.0  0.7  0.7  1.0      1
```

接著我們產生兩組資料, 每組資料有 600筆資料，2個特徵 `X: 600x2` 以及2個類別 `y:600 (前300個元素爲0，餘下爲1)` ：

1. `dataset_fixed_cov()` :2個類別的特徵具備有相同共變數(covariance)
2. `dataset_fixed_cov()` :2個類別的特徵具備有不同之共變數 差異落在X資料的產生 `np.dot(np.random.randn(n, dim), C)` 與 `np.dot(np.random.randn(n, dim), C.T)` 的不同。`np.dot(np.random.randn(n, dim), C)` 會產生 `300x2` 之矩陣，其亂數產生的範圍可交由 `C` 矩陣來控制。在 `dataset_fixed_cov()` 中，前後300筆資料產生之範圍皆由 `C` 來調控。我們可以在最下方的結果圖示看到上排影像(相同共變數)的資料分佈無論是紅色(代表類別1)以及藍色(代表類別2)其分佈形狀相似。而下排影像(不同共變數)，分佈形狀則不同。圖示中，橫軸及縱軸分別表示第一及第二個特徵，讀者可以試著將 `0.83` 這個數字減少或是將 `C.T` 改成 `C` ，看看最後結果圖形有了什麼改變？

```python
def dataset_fixed_cov():
    '''Generate 2 Gaussians samples with the same covariance matrix'''
    n, dim = 300, 2
    np.random.seed(0)
    C = np.array([[0., -0.23], [0.83, .23]])
    X = np.r_[np.dot(np.random.randn(n, dim), C),
              np.dot(np.random.randn(n, dim), C) + np.array([1, 1])] #利用 + np.array([1, 1]) 產
生類別間的差異
    y = np.hstack((np.zeros(n), np.ones(n))) #產生300個零及300個1並連接起來
    return X, y

def dataset_cov():
    '''Generate 2 Gaussians samples with different covariance matrices'''
    n, dim = 300, 2
    np.random.seed(0)
    C = np.array([[0., -1.], [2.5, .7]]) * 2.
    X = np.r_[np.dot(np.random.randn(n, dim), C),
              np.dot(np.random.randn(n, dim), C.T) + np.array([1, 4])]
    y = np.hstack((np.zeros(n), np.ones(n)))
    return X, y
```

(二)繪圖函式

1. 找出 True Positive及False Negative 之辨認點
2. 以紅色及藍色分別表示分類爲 0及1的資料點，而以深紅跟深藍來表示誤判資料
3. 以 `lda.predict_proba()` 畫出分類的機率分佈(請參考範例三)

(爲了方便在ipython notebook環境下顯示，下面函式有經過微調)

```
def plot_data(lda, X, y, y_pred, fig_index):
    splot = plt.subplot(2, 2, fig_index)
    if fig_index == 1:
        plt.title('Linear Discriminant Analysis',fontsize=28)
        plt.ylabel('Data with fixed covariance',fontsize=28)
    elif fig_index == 2:
        plt.title('Quadratic Discriminant Analysis',fontsize=28)
    elif fig_index == 3:
        plt.ylabel('Data with varying covariances',fontsize=28)

    # 步驟一：找出 True Positive及False postive 之辨認點

    tp = (y == y_pred)  # True Positive
    tp0, tp1 = tp[y == 0], tp[y == 1] #tp0 代表分類爲0且列爲 True Positive之資料點
    X0, X1 = X[y == 0], X[y == 1]
    X0_tp, X0_fp = X0[tp0], X0[~tp0]
    X1_tp, X1_fp = X1[tp1], X1[~tp1]

    # 步驟二：以紅藍來畫出分類資料，以深紅跟深藍來表示誤判資料

    # class 0: dots
    plt.plot(X0_tp[:, 0], X0_tp[:, 1], 'o', color='red')
    plt.plot(X0_fp[:, 0], X0_fp[:, 1], '.', color='#990000')  # dark red

    # class 1: dots
    plt.plot(X1_tp[:, 0], X1_tp[:, 1], 'o', color='blue')
    plt.plot(X1_fp[:, 0], X1_fp[:, 1], '.', color='#000099')  # dark blue

    #步驟三：畫出分類的機率分佈(請參考範例三)
    # class 0 and 1 : areas
    nx, ny = 200, 100
    x_min, x_max = plt.xlim()
    y_min, y_max = plt.ylim()
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),
                         np.linspace(y_min, y_max, ny))
    Z = lda.predict_proba(np.c_[xx.ravel(), yy.ravel()])
    Z = Z[:, 1].reshape(xx.shape)
    plt.pcolormesh(xx, yy, Z, cmap='red_blue_classes',
                   norm=colors.Normalize(0., 1.))
    plt.contour(xx, yy, Z, [0.5], linewidths=2., colors='k')

    # means
    plt.plot(lda.means_[0][0], lda.means_[0][1],
             'o', color='black', markersize=10)
    plt.plot(lda.means_[1][0], lda.means_[1][1],
             'o', color='black', markersize=10)

    return splot
```

```
def plot_ellipse(splot, mean, cov, color):
    v, w = linalg.eigh(cov)
    u = w[0] / linalg.norm(w[0])
    angle = np.arctan(u[1] / u[0])
    angle = 180 * angle / np.pi  # convert to degrees
    # filled Gaussian at 2 standard deviation
    ell = mpl.patches.Ellipse(mean, 2 * v[0] ** 0.5, 2 * v[1] ** 0.5,
                              180 + angle, color=color)
    ell.set_clip_box(splot.bbox)
    ell.set_alpha(0.5)
    splot.add_artist(ell)
    splot.set_xticks(())
    splot.set_yticks(())
```

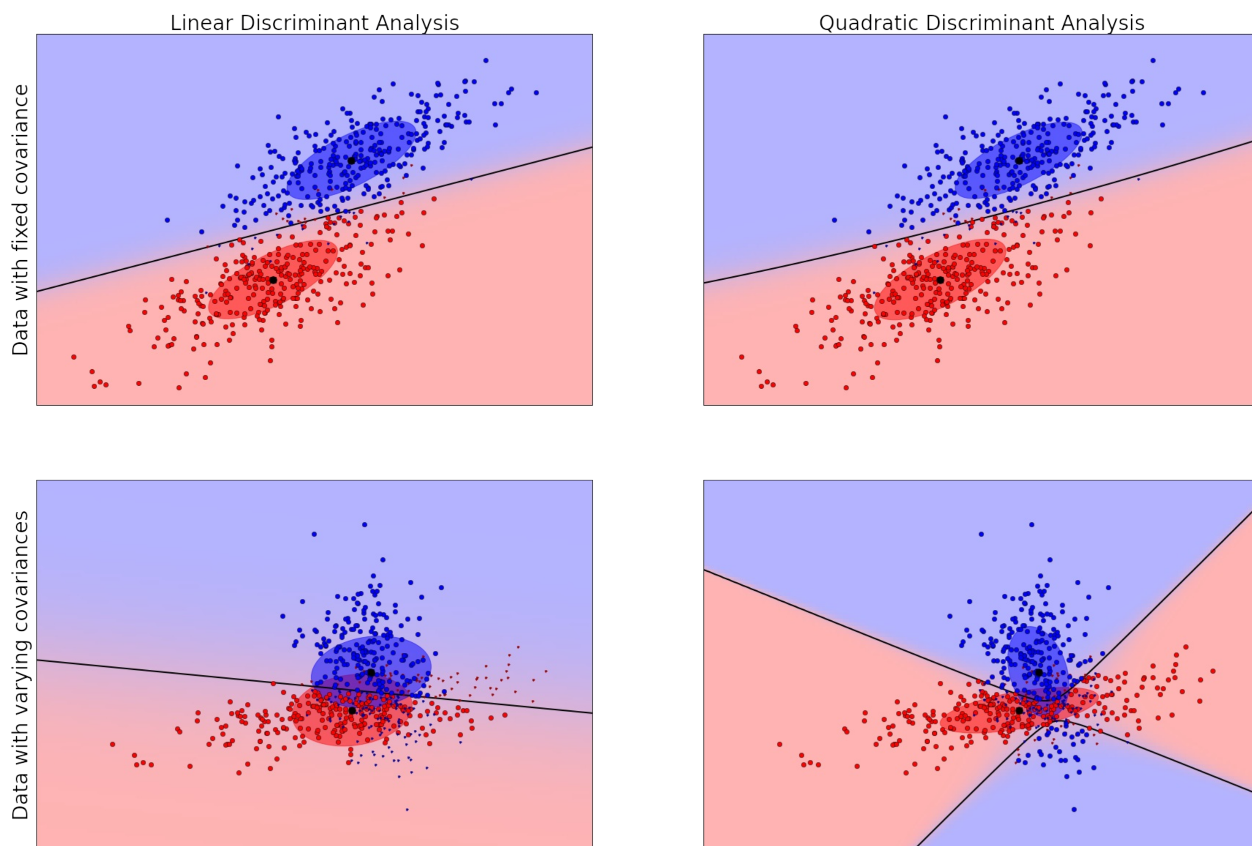
(三)測試資料並繪圖

```
def plot_lda_cov(lda, splot):
    plot_ellipse(splot, lda.means_[0], lda.covariance_, 'red')
    plot_ellipse(splot, lda.means_[1], lda.covariance_, 'blue')


def plot_qda_cov(qda, splot):
    plot_ellipse(splot, qda.means_[0], qda.covariances_[0], 'red')
    plot_ellipse(splot, qda.means_[1], qda.covariances_[1], 'blue')

###############################################################################
figure = plt.figure(figsize=(30,20), dpi=300)
for i, (X, y) in enumerate([dataset_fixed_cov(), dataset_cov()]):
    # Linear Discriminant Analysis
    lda = LinearDiscriminantAnalysis(solver="svd", store_covariance=True)
    y_pred = lda.fit(X, y).predict(X)
    splot = plot_data(lda, X, y, y_pred, fig_index=2 * i + 1)
    plot_lda_cov(lda, splot)
    plt.axis('tight')

    # Quadratic Discriminant Analysis
    qda = QuadraticDiscriminantAnalysis(store_covariances=True)
    y_pred = qda.fit(X, y).predict(X)
    splot = plot_data(qda, X, y, y_pred, fig_index=2 * i + 2)
    plot_qda_cov(qda, splot)
    plt.axis('tight')
plt.suptitle('Linear Discriminant Analysis vs Quadratic Discriminant Analysis',fontsize=28)
plt.show()
```



Python source code: plot_lda_qda.py

http://scikit-learn.org/stable/_downloads/plot_lda_qda.py

```
print(__doc__)

from scipy import linalg
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import colors

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

###############################################################################
# colormap
cmap = colors.LinearSegmentedColormap(
    'red_blue_classes',
    {'red': [(0, 1, 1), (1, 0.7, 0.7)],
     'green': [(0, 0.7, 0.7), (1, 0.7, 0.7)],
     'blue': [(0, 0.7, 0.7), (1, 1, 1)]})
plt.cm.register_cmap(cmap=cmap)


###############################################################################
# generate datasets
def dataset_fixed_cov():
```

```
    '''Generate 2 Gaussians samples with the same covariance matrix'''
    n, dim = 300, 2
    np.random.seed(0)
    C = np.array([[0., -0.23], [0.83, .23]])
    X = np.r_[np.dot(np.random.randn(n, dim), C),
              np.dot(np.random.randn(n, dim), C) + np.array([1, 1])]
    y = np.hstack((np.zeros(n), np.ones(n)))
    return X, y


def dataset_cov():
    '''Generate 2 Gaussians samples with different covariance matrices'''
    n, dim = 300, 2
    np.random.seed(0)
    C = np.array([[0., -1.], [2.5, .7]]) * 2.
    X = np.r_[np.dot(np.random.randn(n, dim), C),
              np.dot(np.random.randn(n, dim), C.T) + np.array([1, 4])]
    y = np.hstack((np.zeros(n), np.ones(n)))
    return X, y


###############################################################################
# plot functions
def plot_data(lda, X, y, y_pred, fig_index):
    splot = plt.subplot(2, 2, fig_index)
    if fig_index == 1:
        plt.title('Linear Discriminant Analysis')
        plt.ylabel('Data with fixed covariance')
    elif fig_index == 2:
        plt.title('Quadratic Discriminant Analysis')
    elif fig_index == 3:
        plt.ylabel('Data with varying covariances')

    tp = (y == y_pred)  # True Positive
    tp0, tp1 = tp[y == 0], tp[y == 1]
    X0, X1 = X[y == 0], X[y == 1]
    X0_tp, X0_fp = X0[tp0], X0[~tp0]
    X1_tp, X1_fp = X1[tp1], X1[~tp1]

    # class 0: dots
    plt.plot(X0_tp[:, 0], X0_tp[:, 1], 'o', color='red')
    plt.plot(X0_fp[:, 0], X0_fp[:, 1], '.', color='#990000')  # dark red

    # class 1: dots
    plt.plot(X1_tp[:, 0], X1_tp[:, 1], 'o', color='blue')
    plt.plot(X1_fp[:, 0], X1_fp[:, 1], '.', color='#000099')  # dark blue

    # class 0 and 1 : areas
    nx, ny = 200, 100
    x_min, x_max = plt.xlim()
    y_min, y_max = plt.ylim()
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),
                         np.linspace(y_min, y_max, ny))
```

```
    Z = lda.predict_proba(np.c_[xx.ravel(), yy.ravel()])
    Z = Z[:, 1].reshape(xx.shape)
    plt.pcolormesh(xx, yy, Z, cmap='red_blue_classes',
                   norm=colors.Normalize(0., 1.))
    plt.contour(xx, yy, Z, [0.5], linewidths=2., colors='k')

    # means
    plt.plot(lda.means_[0][0], lda.means_[0][1],
             'o', color='black', markersize=10)
    plt.plot(lda.means_[1][0], lda.means_[1][1],
             'o', color='black', markersize=10)

    return splot


def plot_ellipse(splot, mean, cov, color):
    v, w = linalg.eigh(cov)
    u = w[0] / linalg.norm(w[0])
    angle = np.arctan(u[1] / u[0])
    angle = 180 * angle / np.pi  # convert to degrees
    # filled Gaussian at 2 standard deviation
    ell = mpl.patches.Ellipse(mean, 2 * v[0] ** 0.5, 2 * v[1] ** 0.5,
                              180 + angle, color=color)
    ell.set_clip_box(splot.bbox)
    ell.set_alpha(0.5)
    splot.add_artist(ell)
    splot.set_xticks(())
    splot.set_yticks(())


def plot_lda_cov(lda, splot):
    plot_ellipse(splot, lda.means_[0], lda.covariance_, 'red')
    plot_ellipse(splot, lda.means_[1], lda.covariance_, 'blue')


def plot_qda_cov(qda, splot):
    plot_ellipse(splot, qda.means_[0], qda.covariances_[0], 'red')
    plot_ellipse(splot, qda.means_[1], qda.covariances_[1], 'blue')

###############################################################################
for i, (X, y) in enumerate([dataset_fixed_cov(), dataset_cov()]):
    # Linear Discriminant Analysis
    lda = LinearDiscriminantAnalysis(solver="svd", store_covariance=True)
    y_pred = lda.fit(X, y).predict(X)
    splot = plot_data(lda, X, y, y_pred, fig_index=2 * i + 1)
    plot_lda_cov(lda, splot)
    plt.axis('tight')

    # Quadratic Discriminant Analysis
    qda = QuadraticDiscriminantAnalysis(store_covariances=True)
    y_pred = qda.fit(X, y).predict(X)
    splot = plot_data(qda, X, y, y_pred, fig_index=2 * i + 2)
    plot_qda_cov(qda, splot)
```

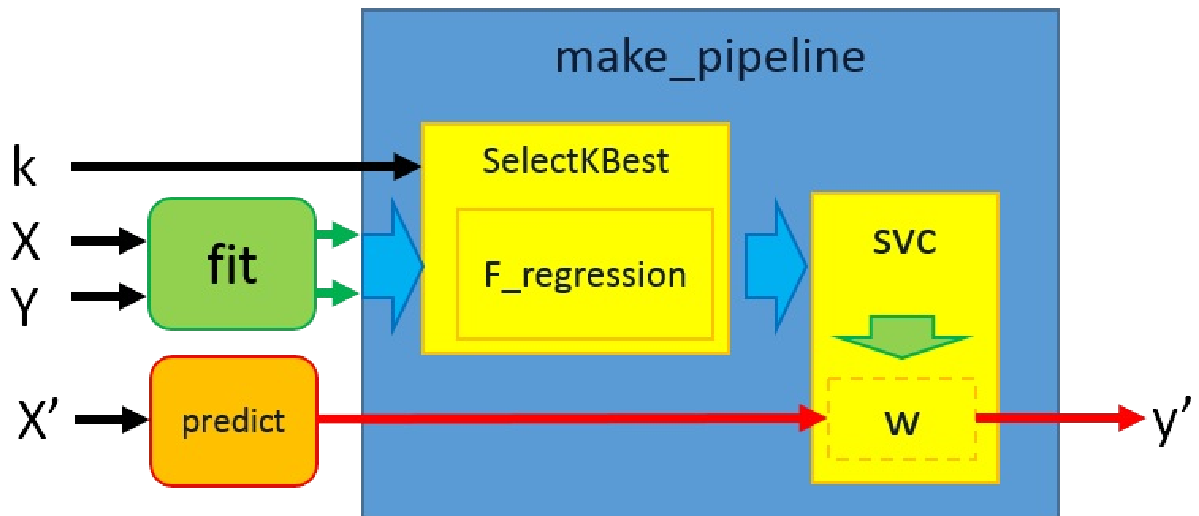
```
    plt.axis('tight')
plt.suptitle('Linear Discriminant Analysis vs Quadratic Discriminant Analysis')
plt.show()
```

Feature Selection

特徵選擇 特徵選擇主要是以統計特徵與目標的相關性、或以疊代排序特徵影響目標影響力的方式來逐漸排除與目標較不相關的特徵，留下與目標最相近的特徵，使判斷準確率能夠提升。

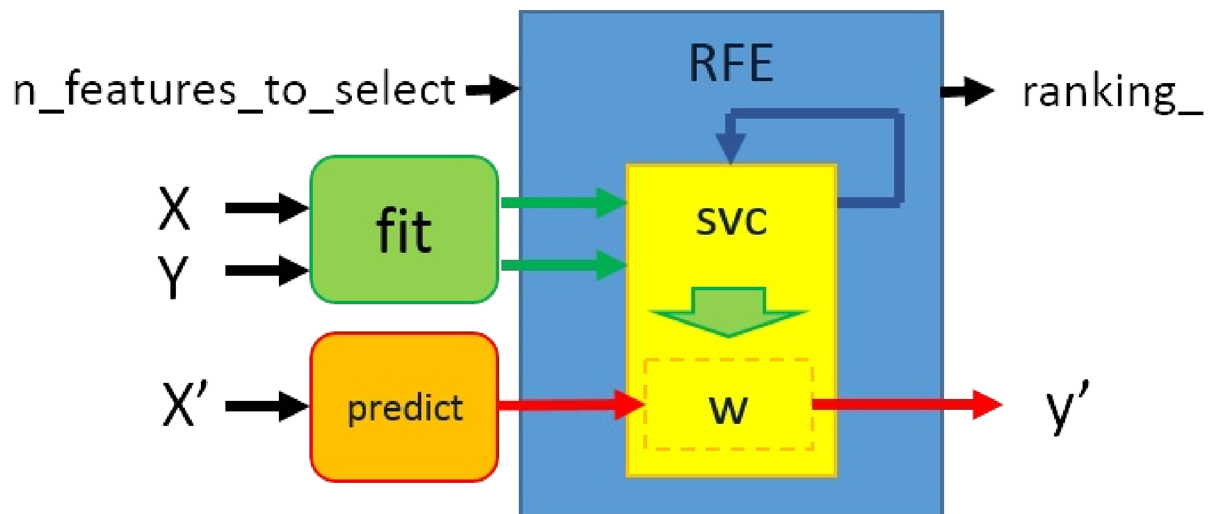
範例一：Pipeline Anova SVM

以anova filter作爲選擇特徵的依據，並示範以傳遞(Pipeline)的方式來執行特徵選擇的訓練。



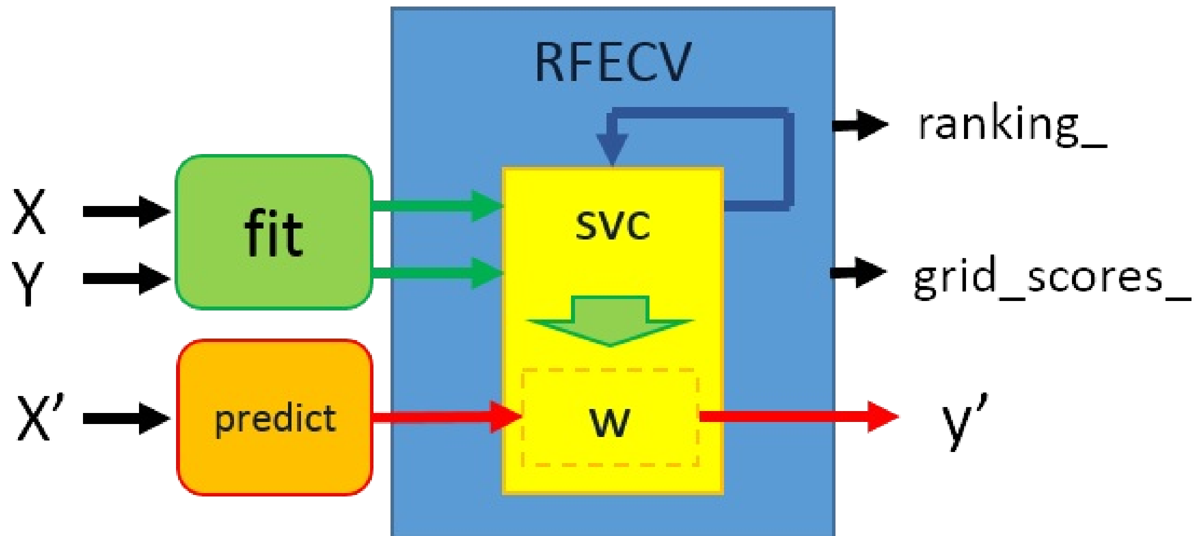
範例二:Recursive feature elimination

以重複排除最不具有特徵影響力的特徵，來減少訓練的特徵數目，直到指定的特徵數目。



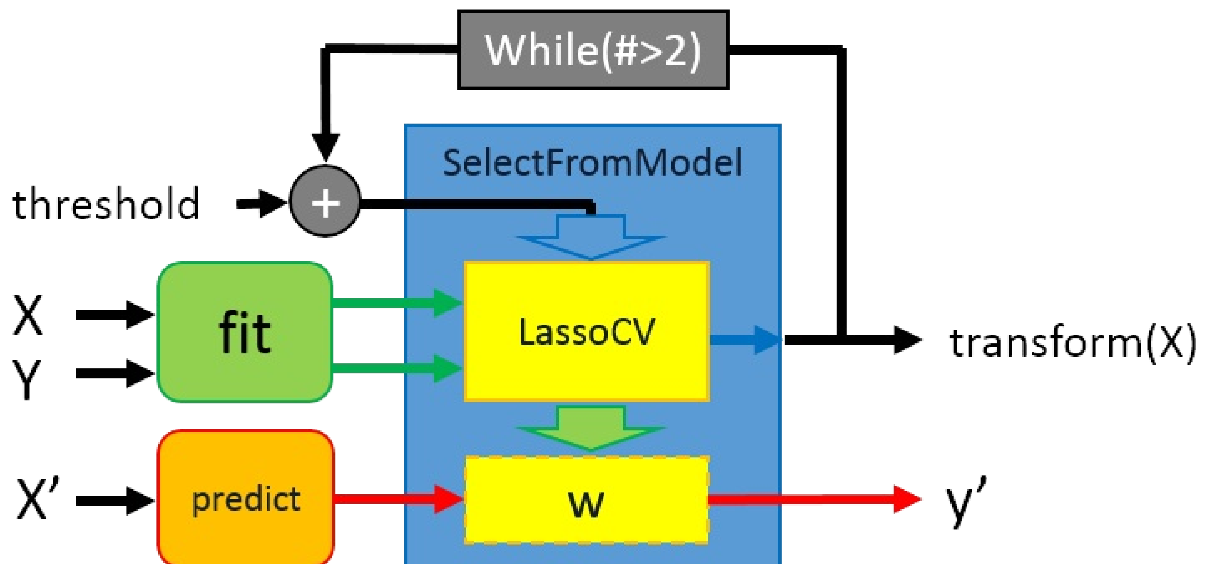
範例三:Recursive feature elimination with cross-validation

除了重複排除不具影響力的特徵外，對每次排除特徵後計算準確度，以準確度最高的特徵數目作爲選定訓練特徵數目的依據。



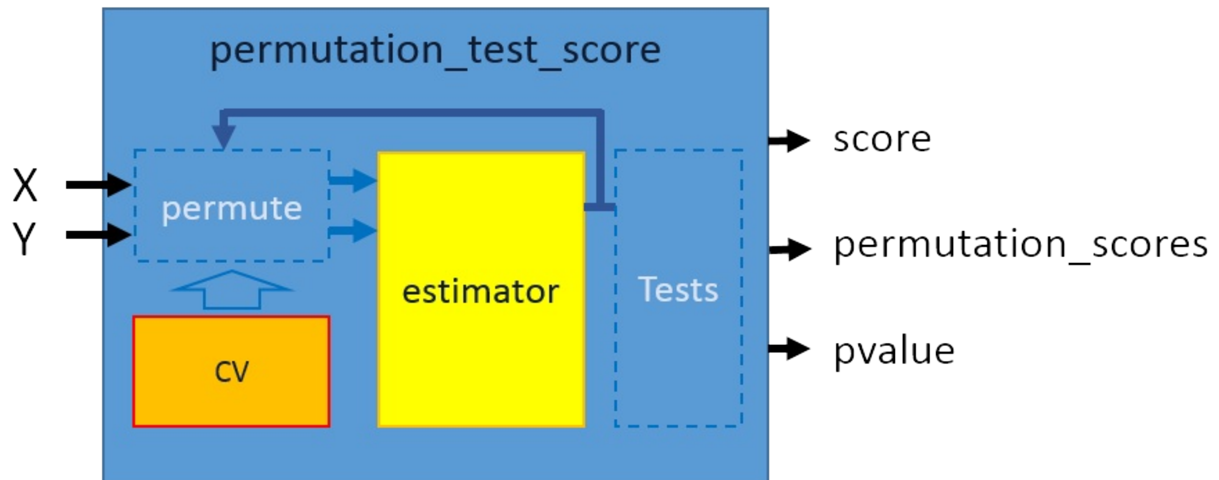
範例四:Feature selection using SelectFromModel and LassoCV

示範如何使用SelectFromModel函式來選擇給定的函式，並設置輸入函式的門檻值，用以判斷訓練的特徵數目。在本範例是使用LassoCV作爲選擇的涵式。



範例五:Test with permutations the significance of a classification score

本範例示範當目標類型的特徵，並無數值的大小區分時，以置換分類目標的方式來找到最高準確率的特徵挑選結果。以避免因爲特徵目標分類轉換爲用以區分不同類型時造成的誤判。



範例六:Univariate Feature Selection

本範例示範用SelectPercentile以統計的方式來做特徵的選擇，並比指定的判斷函式來挑選特徵。本範例的輸入涵式爲ANOVA，並以計算的F-value來做爲挑選特徵的判斷。

特徵選擇/範例一: Pipeline Anova SVM

http://scikit-learn.org/stable/auto_examples/feature_selection/feature_selection_pipeline.html

此機器學習範例示範pipeline的使用，依照順序執行ANOVA挑選主要特徵，並且使用C-SVM來計算特徵的權重與預測。

1. 使用 `make_classification` 建立模擬資料
2. 使用 `SelectKBest` 設定要用哪種目標函式，以挑出可提供信息的特徵
3. 使用 `SVC` 設定支持向量機爲分類計算以及其核函數
4. 用 `make_pipeline` 合併 SelectKBest物件 與 SVC物件
5. 用 `fit` 做訓練，並且以 `predict` 來做預測

(一)建立模擬資料

在選擇特徵之前需要有整理好的特徵與目標資料。在此範例中，將以 `make_classification` 功能建立特徵與目標。該功能可以依照使用者想模擬的情況，建立含有不同特性的模擬資料，像是總特徵數目，其中有幾項特徵含有目標資訊性、目標聚集的程度、目標分爲幾類等等的特性。

```
# import some data to play with
X, y = samples_generator.make_classification(
    n_features=20, n_informative=3, n_redundant=0, n_classes=4,
    n_clusters_per_class=2)
```

在本範例，我們將X建立爲一個有20個特徵的資料，其中有3種特徵具有目標資訊性，0個特徵是由目標資訊性特徵所產生的線性組合，目標分爲4類，而每個分類的目標分布爲2個群集。

(二)選擇最好的特徵

在機器學習的訓練之前，可以藉由統計或指定評分函數，算出特徵與目標之間的關係，並挑選出最具有關係的特徵作爲訓練的素材，而不直接使用所有特徵做爲訓練的素材。

其中一種方法是統計特徵與目標之間的F-score做爲評估分數，再挑選F-score最高的幾個特徵作爲訓練素材。我們可以用 `SelectKBest()` 來建立該功能的運算物件。

```
# ANOVA SVM-C
# 1) anova filter, take 3 best ranked features
anova_filter = SelectKBest(f_regression, k=3)
```

`SelectKBest()` 的第一項參數須給定評分函數，在本範例是設定爲 `f_regression` 。第二項參數代表選擇評估分數最高的3個特徵做爲訓練的素材。建立完成後，即可用物件內的方法 `.fit_transform(X,y)` 來提取被選出來的特徵。

(三)以pipeline方式來設定支持向量機分類法運算物件

Scikit-lenarn的支持向量機分類函式庫提供使用簡單易懂的指令，只要用 `SVC()` 建立運算物件後，便可以用運算物件內的方法 `.fit()` 與 `.predict()` 來做訓練與預測。

本範例在建立運算物件後，不直接用 `SelectKBest().fit_transform()` 提出訓練素材。而是以 `make_pipeline()` 合併先前設定好的兩個運算物件。再執行 `.fit()` 與 `.predict()` 來完成訓練與預測的動作。

```
# 2) svm
clf = svm.SVC(kernel='linear')

anova_svm = make_pipeline(anova_filter, clf)
anova_svm.fit(X, y)
anova_svm.predict(X)
```

當我們以佇列建立好的運算物件，就可以直接給定所有的特徵資料與目標資料做訓練與預測。在訓練過程中，會依照給定的特徵素材數目從特徵資料中挑出特徵素材。預測時，也會從預測資料中挑出對應特徵素材的資料來做預測判斷。

若是將 `SelectKBest()` 與 `SVC()` 物件分開來執行，當 `SVC()` 物件在做學習時給定的特徵即爲被選出來的特徵素材數目。那預測的時候也必須從預測資料中，挑出被 `SelectKBest()` 選出來的特徵來給 `SVC()` 做預測。

(四)原始碼

Python source code: feature_selection_pipeline.py

```
from sklearn import svm
from sklearn.datasets import samples_generator
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import make_pipeline

# import some data to play with
X, y = samples_generator.make_classification(
    n_features=20, n_informative=3, n_redundant=0, n_classes=4,
    n_clusters_per_class=2)

# ANOVA SVM-C
# 1) anova filter, take 3 best ranked features
anova_filter = SelectKBest(f_regression, k=3)
# 2) svm
clf = svm.SVC(kernel='linear')

anova_svm = make_pipeline(anova_filter, clf)
anova_svm.fit(X, y)
anova_svm.predict(X)
```

(五)函式用法

`make_classification()` 的參數

```
sklearn.datasets.make_classification(   n_samples=100,
                                        n_features=20,
                                        n_informative=2,
                                        n_redundant=2,
                                        n_repeated=0,
                                        n_classes=2,
                                        n_clusters_per_class=2,
                                        weights=None,
                                        flip_y=0.01,
                                        class_sep=1.0,
                                        hypercube=True,
                                        shift=0.0,
                                        scale=1.0,
                                        shuffle=True,
                                        random_state=None)
```

參數:

- n_samples :
- n_fratures : 總特徵數目
- n_informative: 有意義的特徵數目
- n_redundant : 產生有意義特徵的隨機線性組合
- n_repeated
- n_classes: 共分類爲幾類
- n_clusters_per_class: 一個類群有幾個群組分布
- weights :
- flip_y :
- class_sep :
- hypercube :
- shift :
- scale :
- shuffle :
- random_state :

輸出:

- X : 特徵矩陣資料
- Y : 對應目標資料

類似的功能:

make_blobs

make_gaussian_quantiles

`SelectKBest()` 的參數

SelectKBest 的使用:

- 選擇最好的特徵(目標函式, 特徵個數)
- 目標函式: 測試X與Y之間關係，須提供F score與p-value
- 特徵個數: 最好的特徵個數

f_regression 的使用 :

- f_regression(X,y)
- 輸入X與y
- 輸出F score與p-value

特徵選擇/範例二: Recursive feature elimination

http://scikit-learn.org/stable/auto_examples/feature_selection/plot_rfe_digits.html

本範例主要目的是減少特徵數量來提升機器學習之預測準確度。 主要方法是去不斷去剔除與資料分類關係轉少之特徵，來篩選特徵數目至指定數目。

1. 以 `load_digits` 取得內建的數字辨識資料
2. 以 `RFE` 疊代方式刪去相對不具有目標影響力的特徵.

(一)產生內建的數字辨識資料

```
# Load the digits dataset
digits = load_digits()
X = digits.images.reshape((len(digits.images), -1))
y = digits.target
```

數位數字資料是解析度為8*8的手寫數字影像，總共有1797筆資料。預設為0~9十種數字類型，亦可由n_class來設定要取得多少種數字類型。

輸出的資料包含

1. ‘data’, 特徵資料(1797*64)
2. ‘images’, 影像資料(1797*8*8)
3. ‘target’, 資料標籤(1797)
4. ‘target_names’, 選取出的標籤列表(與n_class給定的長度一樣)
5. ‘DESCR’, 此資料庫的描述

可以參考Classification的Ex1

(二)以疊代方式計算模型

`RFE` 以排除最不具目標影響力的特徵，做特徵的影響力排序。並且將訓練用的特徵挑選至 `n_features_to_select` 所給定的特徵數。因為要看每一個特徵的影響力排序，所以我們將 `n_features_to_select` 設定為1，一般會根據你所知道的具有影響力特徵數目來設定該參數。而 `step` 代表每次刪除較不具影響力的特徵數目，因為本範例要觀察每個特徵的影響力排序，所以也是設定為1。若在實際應用時，特徵的數目較大，可以考慮將 `step` 的參數設高一點。

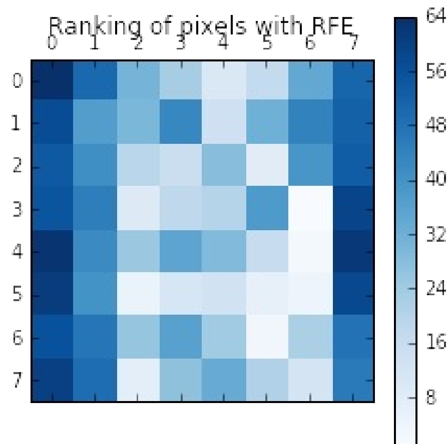
```
# Create the RFE object and rank each pixel
svc = SVC(kernel="linear", C=1)
rfe = RFE(estimator=svc, n_features_to_select=1, step=1)
rfe.fit(X, y)
ranking = rfe.ranking_.reshape(digits.images[0].shape)
```

可以用方法 `ranking_` 來看輸入的特徵權重關係。而方法 `estimator_` 可以取得訓練好的分類機狀態。比較特別的是當我們核函數是以線性來做分類時， `estimator_` 下的方法 `coef_` 即為特徵的分類權重矩陣。權重矩陣的大小會因為 `n_features_to_select` 與資料的分類類別而改變，譬如本範例是十個數字的分類，並選擇以一個特徵來做分類訓練，就會得到45*1的係數矩陣，其中45是從分類類別所需要的判斷式而來，與巴斯卡三角形的第三層數正比。

(三)畫出每個像素所對應的權重順序

取得每個像素位置對於判斷數字的權重順序後，我們把權重順序依照顏色畫在對應的位置，數值愈大代表該像素是較不重要之特徵。由結果來看，不重要之特徵多半位於影像之外圍部份。而所有的訓練影像中，外圍像素多半為空白，因此較不重要。

```
# Plot pixel ranking
plt.matshow(ranking, cmap=plt.cm.Blues)
plt.colorbar()
plt.title("Ranking of pixels with RFE")
plt.show()
```



(四)原始碼

Python source code: plot_rfe_digits.py

```
print(__doc__)

from sklearn.svm import SVC
from sklearn.datasets import load_digits
from sklearn.feature_selection import RFE
import matplotlib.pyplot as plt

# Load the digits dataset
digits = load_digits()
X = digits.images.reshape((len(digits.images), -1))
y = digits.target

# Create the RFE object and rank each pixel
svc = SVC(kernel="linear", C=1)
rfe = RFE(estimator=svc, n_features_to_select=1, step=1)
rfe.fit(X, y)
ranking = rfe.ranking_.reshape(digits.images[0].shape)

# Plot pixel ranking
plt.matshow(ranking, cmap=plt.cm.Blues)
plt.colorbar()
plt.title("Ranking of pixels with RFE")
plt.show()
```

特徵選擇/範例三: Recursive feature elimination with cross-validation

http://scikit-learn.org/stable/auto_examples/feature_selection/plot_rfe_with_cross_validation.html

REFCV比REF多一個交叉比對的分數(grid*scores*)，代表選擇多少特徵後的準確率。但REFCV不用像REF要給定選擇多少特徵，而是會依照交叉比對的分數而自動選擇訓練的特徵數。

本範例示範 `RFE` 的進階版，當我們在使用 `RFE` 指令時，需要輸入訓練特徵數目，讓訓練機能排除到其他較不具有影響力的特徵，也就是要有預期的訓練特徵數目。在 `RFECV` 指令提供了使用交叉驗證來選擇有最好準確率的訓練特徵數目。而交叉驗證也可以幫助我們避免訓練時造成過度訓練(overfitting)的現象，也就是當我們從某一組資料中挑出一筆訓練資料，能夠對剩下的測試資料預測出準確度最好的分類，卻發現這個分類機狀態無法準確的辨識新進資料的結果，因爲這個最佳狀態只適用在特定的組合情況。因此使用 `RFECV` 後，我們可以從結果看出，使用多少特徵做分類判斷可以得到的準確率高低。

1. 以疊代方式計算模型
2. 以交叉驗證來取得影響力特徵

(一)建立模擬資料

```
# Build a classification task using 3 informative features
X, y = make_classification(n_samples=1000, n_features=25, n_informative=3,
                           n_redundant=2, n_repeated=0, n_classes=8,
                           n_clusters_per_class=1, random_state=0)
```

說明可以參考EX1，執行過此函數後，我們可以得到具有25個特徵且有1000樣本的資料，其中具有目標影響力的特徵有三個，有兩個特徵是由具有資訊影響力的特徵線性組合出來的，而目標共有八個分類類別。

(二)以疊代排序特徵影響力，並以交叉驗證來選出具有實際影響力的特徵

在使用 `RFECV` 指令前，需要建立支持向量機物件，以及交叉驗證的形式。本範例仍使用 `SVC` 以及線性核函數來作爲主要的分類機。

在交叉驗證的部分，我們使用 `StratifiedKFold` 指令來做K 堆疊(Fold)的交叉驗證。也就是將資料分爲K堆，一堆作爲預測用，剩下的(K-1)堆則用來訓練，經過計算後，再以另外一堆作爲預測，重複K次。

而 `scoring` 參數則是依照分類資料的形式，輸入對應的評分方式。以本例子爲超過兩類型的分類，因此使用'accuracy'來對多重分類的評分方式。詳細可參考scoring

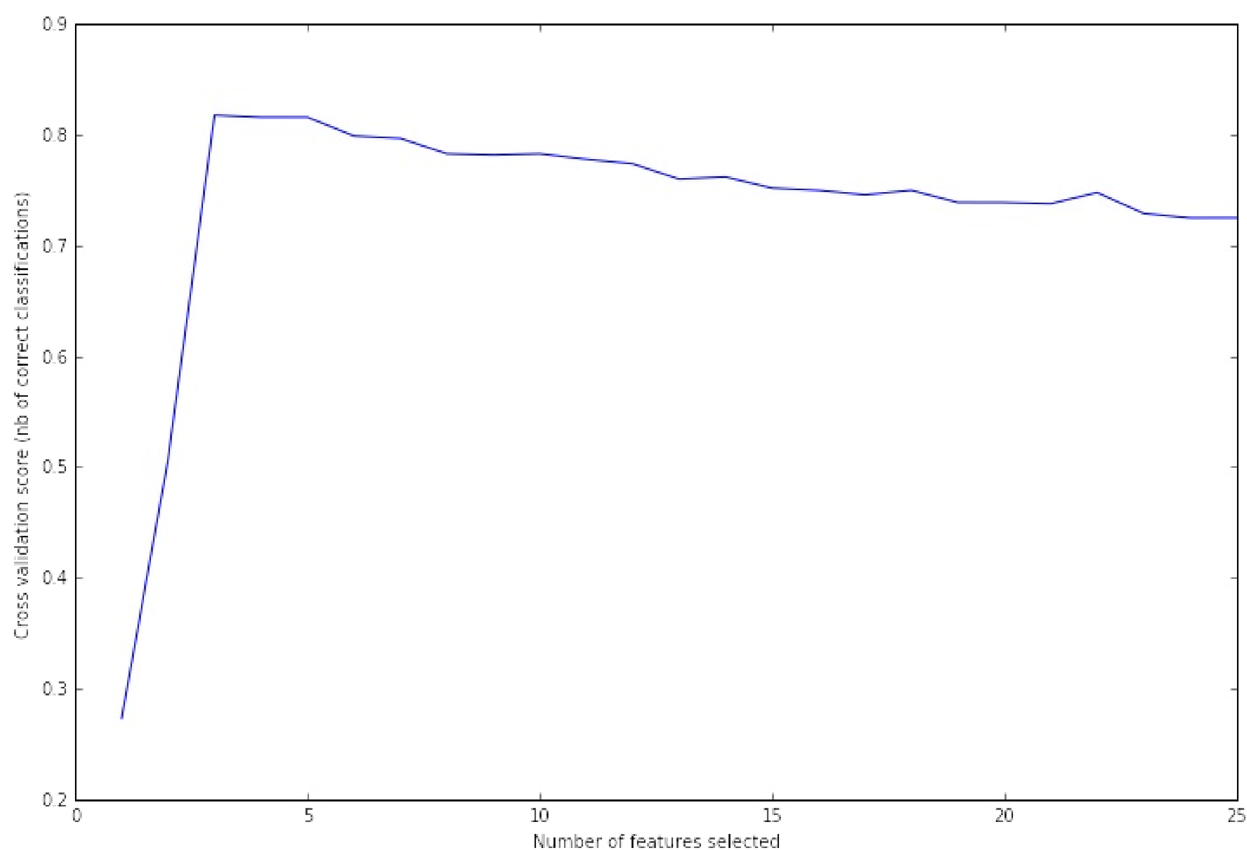
```
# Create the RFE object and compute a cross-validated score.
svc = SVC(kernel="linear")
# The "accuracy" scoring is proportional to the number of correct
# classifications
rfecv = RFECV(estimator=svc, step=1, cv=StratifiedKFold(y, 2),
              scoring='accuracy')
rfecv.fit(X, y)

print("Optimal number of features : %d" % rfecv.n_features_)
```

以RFECV設定好的功能物件，即可用以做訓練的動作。其結果可由n*features*得知有幾樣特徵是具有實際影響力。並可以由grid*scores*看出特徵的多寡如何影響準確率。 此功能需要設定交叉驗證的形式，本範例是以交叉驗證產生器做爲輸入，其功能介紹如下。

(三)畫出具有影響力特徵對應準確率的圖

下圖的曲線表示選擇多少個特徵來做訓練，會得到多少的準確率。



可以看到選擇三個最具有影響力的特徵時，交叉驗證的準確率高達81.8%。與建立模擬資料的n_informative=3是相對應的。

(四) 原始碼出處

Python source code: plot_rfe_digits.py

```
print(__doc__)

import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold
from sklearn.feature_selection import RFECV
from sklearn.datasets import make_classification

# Build a classification task using 3 informative features
X, y = make_classification(n_samples=1000, n_features=25, n_informative=3,
                           n_redundant=2, n_repeated=0, n_classes=8,
                           n_clusters_per_class=1, random_state=0)

# Create the RFE object and compute a cross-validated score.
svc = SVC(kernel="linear")
# The "accuracy" scoring is proportional to the number of correct
# classifications
rfecv = RFECV(estimator=svc, step=1, cv=StratifiedKFold(y, 2),
              scoring='accuracy')
rfecv.fit(X, y)

print("Optimal number of features : %d" % rfecv.n_features_)

# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```

本章介紹到函式用法

`RFECV()` 的參數

```
class sklearn.feature_selection.RFECV(estimator, step=1, cv=None, scoring=None, estimator_param
s=None, verbose=0)[source]
```

參數

- estimator
- step
- cv: 若無輸入，預設爲3-fold的交叉驗證。輸入整數i，則做i-fold交叉驗證。若爲物件，則以該物件做爲交叉驗證產生器。
- scoring
- estimator_params
- verbose

輸出

- n_features_: 預測有影響力的特徵的總數目
- support_: 有影響力的特徵遮罩，可以用來挑出哪些特徵
- ranking_: 各特徵的影響力程度
- grid*scores*: 從最有影響力的特徵開始加入，計算使用多少個特徵對應得到的準確率。

- estimator_

特徵選擇/範例四: Feature selection using SelectFromModel and LassoCV

http://scikit-learn.org/stable/auto_examples/feature_selection/plot_select_from_model_boston.html

此範例是示範以 `LassoCV` 來挑選特徵，Lasso是一種用來計算稀疏矩陣的線性模形。在某些情況下是非常有用的，因為在此演算過程中會以較少數的特徵來找最佳解，基於參數有相依性的情況下，使變數的數目有效的縮減。因此，Lasso法以及它的變形式可算是壓縮參數關係基本方法。在某些情況下，此方法可以準確的偵測非零權重的值。

Lasso最佳化的目標函數:



1. 以 `LassoCV` 法來計算目標資訊性特徵數目較少的資料
2. 用 `SelectFromModel` 設定特徵重要性的門檻值來選擇特徵
3. 提高 `SelectFromModel` 的 `.threshold` 使目標資訊性特徵數逼近預期的數目

(一)取得波士頓房產資料

```
from sklearn.datasets import load_boston
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LassoCV

# Load the boston dataset.
boston = load_boston()
X, y = boston['data'], boston['target']
```

(二)使用LassoCV功能來篩選具有影響力的特徵

1. 由於資料的類型為連續數字，選用LassoCV來做最具有代表性的特徵選取。
2. 當設定好門檻值，並做訓練後，可以用transform(X)取得計算過後，被認為是具有影響力的特徵以及對應的樣本，可以由其列的數目知道總影響力特徵有幾個。
3. 後面使用了增加門檻值來達到限制最後特徵數目的
4. 使用門檻值來決定後來選取的參數，其說明在下一個標題。
5. 需要用後設轉換

(三)設定選取參數的門檻值

```
while n_features > 2:
    sfm.threshold += 0.1
    X_transform = sfm.transform(X)
    n_features = X_transform.shape[1]
```

(四)原始碼之出處

Python source code: plot_select_from_model_boston.py

```
# Author: Manoj Kumar <mks542@nyu.edu>
# License: BSD 3 clause

print(__doc__)

import matplotlib.pyplot as plt
import numpy as np

from sklearn.datasets import load_boston
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LassoCV

# Load the boston dataset.
boston = load_boston()
X, y = boston['data'], boston['target']

# We use the base estimator LassoCV since the L1 norm promotes sparsity of features.
clf = LassoCV()

# Set a minimum threshold of 0.25
sfm = SelectFromModel(clf, threshold=0.25)
sfm.fit(X, y)
n_features = sfm.transform(X).shape[1]

# Reset the threshold till the number of features equals two.
# Note that the attribute can be set directly instead of repeatedly
# fitting the metatransformer.
while n_features > 2:
    sfm.threshold += 0.1
    X_transform = sfm.transform(X)
    n_features = X_transform.shape[1]

# Plot the selected two features from X.
plt.title(
    "Features selected from Boston using SelectFromModel with "
    "threshold %0.3f." % sfm.threshold)
feature1 = X_transform[:, 0]
feature2 = X_transform[:, 1]
plt.plot(feature1, feature2, 'r.')
plt.xlabel("Feature number 1")
plt.ylabel("Feature number 2")
plt.ylim([np.min(feature2), np.max(feature2)])
plt.show()
```

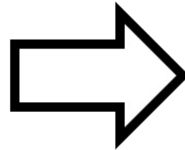
特徵選擇/範例五: Test with permutations the significance of a classification score

http://scikit-learn.org/stable/auto_examples/feature_selection/plot_permutation_test_for_classification.html

此範例主要是用於當我們做機器學習分類時，資料標籤爲無大小關係的分類，也就是第一類與第二類並無前後大小關係的分類。由於輸入分類器的標籤仍爲數值，但數值的大小可能影響分類結果，因此隨機置換分類標籤以及隨機的訓練測試資料組(交叉驗證)來輸入分類機，針對不同類型的分類做對應的評分，統計出不同的資料與標籤組合所得到的準確度與標籤的顯著性。 `permutation_test_score` 提供了對分類標籤做隨機置換的功能，並依照給定的置換次數來計算不同的資料組合配上置換過標籤的組合，用交叉驗證來計算準確性分佈，並統計顯著性。計算過後可取得該分類機器的眞實分數與經過數次組合後取得的分數。

Data set1						
O	X	O	X	O	X	+
O	O	O	X	X	O	+
O	O	X	O	O	X	+
O	X	X	X	X	O	-
X	X	O	X	X	X	-

permutations



Data set2						
O	X	X	X	X	O	+
X	X	O	X	X	X	+
O	O	O	X	X	O	-
O	X	O	X	O	X	-
O	O	O	X	X	O	-

1. 資料集：鳶尾花
2. 特徵：萼片(sepal)之長與寬以及花瓣(petal)之長與寬
3. 預測目標：共有三種鳶尾花 setosa, versicolor, virginica
4. 機器學習方法：線性分類
5. 探討重點：變換訓練資料分類的目標標籤，減少標籤數值對分類的影響
6. 關鍵函式： `sklearn.cross_validation.permutation_test_score`

【1】Ojala and Garriga. Permutation Tests for Studying Classifier Performance. The Journal of Machine Learning Research (2010) vol. 11

(一)取得鳶尾花資料

本範例使用 `datasets.load_iris()` 讀取具有4個資訊影響力特徵與150個樣本的鳶尾花資料，該資料被分類爲三個類型。並且額外增加2200筆150長度的雜訊做爲不具資訊影響力的特徵，來增加辨認複雜度。

```
# Loading a dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target
n_classes = np.unique(y).size

# Some noisy data not correlated
random = np.random.RandomState(seed=0)
E = random.normal(size=(len(X), 2200))

# Add noisy data to the informative features for make the task harder
X = np.c_[X, E]
```

(二)建立基本的支持向量分類機

使用 `SVC` 建立最基本的支持向量分類機。並設定訓練交叉驗證的摺疊系數爲2。

```
svm = SVC(kernel='linear')
cv = StratifiedKFold(y, 2)
```

(三)重複隨機變換訓練資料並統計準確率

當整理好訓練資料，以及支持向量分類機的設定後，我們以 `permutation_test_score` 功能來測試不同的隨機訓練資料組合，以及對應的分類機分數。除了基本的支持向量機物件、訓練資料、訓練目標，還需要指定對分類結果的評分方式、交叉驗證物件。與重複隨機變換法有關的參數像是置換次數(預設爲100)與使用CPU的數目(預設爲1)也可依照使用者使用情況而改變。

```
score, permutation_scores, pvalue = permutation_test_score(
    svm, X, y, scoring="accuracy", cv=cv, n_permutations=100, n_jobs=1)

print("Classification score %s (pvalue : %s)" % (score, pvalue))
```

經過計算的結果，會給予實際的分類機分數、每次隨機置換的分數以及p-value。

(四)統計隨機置換資料算出來的分類機分數圖表

最後一個部分，就是把 `permutation_test_score` 算出來的結果以圖表的方式呈現。

```
###############################################################################
# View histogram of permutation scores
plt.hist(permutation_scores, 20, label='Permutation scores')
ylim = plt.ylim()
# BUG: vlines(..., linestyle='--') fails on older versions of matplotlib
#plt.vlines(score, ylim[0], ylim[1], linestyle='--',
#          color='g', linewidth=3, label='Classification Score'
#          ' (pvalue %s)' % pvalue)
#plt.vlines(1.0 / n_classes, ylim[0], ylim[1], linestyle='--',
#          color='k', linewidth=3, label='Luck')
plt.plot(2 * [score], ylim, '--g', linewidth=3,
         label='Classification Score'
         ' (pvalue %s)' % pvalue)
plt.plot(2 * [1. / n_classes], ylim, '--k', linewidth=3, label='Luck')

plt.ylim(ylim)
plt.legend()
plt.xlabel('Score')
plt.show()
```



原始碼出處

Python source code: plot_select_from_model_boston.py

```
# Author:  Alexandre Gramfort <alexandre.gramfort@inria.fr>
# License: BSD 3 clause

print(__doc__)

import numpy as np
import matplotlib.pyplot as plt

from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold, permutation_test_score
from sklearn import datasets


##############################################################################
# Loading a dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target
n_classes = np.unique(y).size

# Some noisy data not correlated
random = np.random.RandomState(seed=0)
E = random.normal(size=(len(X), 2200))

# Add noisy data to the informative features for make the task harder
X = np.c_[X, E]

svm = SVC(kernel='linear')
cv = StratifiedKFold(y, 2)

score, permutation_scores, pvalue = permutation_test_score(
    svm, X, y, scoring="accuracy", cv=cv, n_permutations=100, n_jobs=1)

print("Classification score %s (pvalue : %s)" % (score, pvalue))

###############################################################################
# View histogram of permutation scores
plt.hist(permutation_scores, 20, label='Permutation scores')
ylim = plt.ylim()
# BUG: vlines(..., linestyle='--') fails on older versions of matplotlib
#plt.vlines(score, ylim[0], ylim[1], linestyle='--',
#          color='g', linewidth=3, label='Classification Score'
#          ' (pvalue %s)' % pvalue)
#plt.vlines(1.0 / n_classes, ylim[0], ylim[1], linestyle='--',
#          color='k', linewidth=3, label='Luck')
plt.plot(2 * [score], ylim, '--g', linewidth=3,
         label='Classification Score'
         ' (pvalue %s)' % pvalue)
plt.plot(2 * [1. / n_classes], ylim, '--k', linewidth=3, label='Luck')

plt.ylim(ylim)
plt.legend()
```

```
plt.xlabel('Score')
plt.show()
```

特徵選擇/範例六: Univariate Feature Selection

http://scikit-learn.org/stable/auto_examples/feature_selection/plot_feature_selection.html

此範例示範單變量特徵的選擇。鳶尾花資料中會加入數個雜訊特徵(不具影響力的特徵資訊)並且選擇單變量特徵。選擇過程會畫出每個特徵的 p-value 與其在支持向量機中的權重。可以從圖表中看出主要影響力特徵的選擇會選出具有主要影響力的特徵，並且這些特徵會在支持向量機有相當大的權重。 在本範例的所有特徵中，只有最前面的四個特徵是對目標有意義的。我們可以看到這些特徵的單變量特徵評分很高。而支持向量機會賦予最主要的權重到這些具影響力的特徵之一，但也會挑選剩下的特徵來做判斷。在支持向量機增加權重之前就確定那些特徵較具有影響力，從而增加辨識率。

1. 資料集：鳶尾花
2. 特徵：萼片(sepal)之長與寬以及花瓣(petal)之長與寬
3. 預測目標：共有三種鳶尾花 setosa, versicolor, virginica
4. 機器學習方法：線性分類
5. 探討重點：使用單變量選擇(`SelectPercentile`)挑出訓練特徵，與直接將所有訓練特徵輸入的分類器做比較
6. 關鍵函式： `sklearn.feature_selection.SelectPercentile`

(一)修改原本的鳶尾花資料

用 `datasets.load_iris()` 讀取鳶尾花的資料做爲具有影響力的特徵，並以 `np.random.uniform` 建立二十個隨機資料做爲不具影響力的特徵，並合併做爲訓練樣本。

```
# import some data to play with

# The iris dataset
iris = datasets.load_iris()

# Some noisy data not correlated
E = np.random.uniform(0, 0.1, size=(len(iris.data), 20))

# Add the noisy data to the informative features
X = np.hstack((iris.data, E))
y = iris.target
```

(二)使用f-value作爲判斷的基準來找主要影響力特徵

以 `SelectPercentile` 作單變量特徵的計算，以F-test(`f_classif`)來做爲選擇的統計方式，挑選函式輸出結果大於百分之十的特徵。並將計算出來的單便量特徵分數結果做正規化，以便比較每特徵在使用單變量計算與未使用單變量計算的差別。

```
###############################################################################
# Univariate feature selection with F-test for feature scoring
# We use the default selection function: the 10% most significant features
selector = SelectPercentile(f_classif, percentile=10)
selector.fit(X, y)
scores = -np.log10(selector.pvalues_)
scores /= scores.max()
plt.bar(X_indices - .45, scores, width=.2,
        label=r'Univariate score ($-Log(p_{value})$)', color='g')
```

(三)找出不計算單變量特徵的分類權重

以所有特徵資料，以線性核函數丟入支持向量分類機，找出各特徵的權重。

```
###############################################################################
# Compare to the weights of an SVM
clf = svm.SVC(kernel='linear')
clf.fit(X, y)

svm_weights = (clf.coef_ ** 2).sum(axis=0)
svm_weights /= svm_weights.max()

plt.bar(X_indices - .25, svm_weights, width=.2, label='SVM weight', color='r')
```

(四)找出以單變量特徵選出的分類權重

以單變量特徵選擇選出的特徵，做爲分類的訓練特徵，差別在於訓練的特徵資料是使用 `selector.transform(X)` 將 `SelectPercentile` 選擇的結果讀取出來，並算出以單變量特徵選擇做預先選擇後，該分類器的判斷權重。

```
clf_selected = svm.SVC(kernel='linear')
clf_selected.fit(selector.transform(X), y)

svm_weights_selected = (clf_selected.coef_ ** 2).sum(axis=0)
svm_weights_selected /= svm_weights_selected.max()

plt.bar(X_indices[selector.get_support()] - .05, svm_weights_selected,
        width=.2, label='SVM weights after selection', color='b')
```

(五)原始碼出處

Python source code: plot_feature_selection.py

```
print(__doc__)

import numpy as np
import matplotlib.pyplot as plt

from sklearn import datasets, svm
from sklearn.feature_selection import SelectPercentile, f_classif

###############################################################################
# import some data to play with

# The iris dataset
iris = datasets.load_iris()

# Some noisy data not correlated
E = np.random.uniform(0, 0.1, size=(len(iris.data), 20))

# Add the noisy data to the informative features
X = np.hstack((iris.data, E))
y = iris.target
```

```
###############################################################################
plt.figure(1)
plt.clf()

X_indices = np.arange(X.shape[-1])

###############################################################################
# Univariate feature selection with F-test for feature scoring
# We use the default selection function: the 10% most significant features
selector = SelectPercentile(f_classif, percentile=10)
selector.fit(X, y)
scores = -np.log10(selector.pvalues_)
scores /= scores.max()
plt.bar(X_indices - .45, scores, width=.2,
        label=r'Univariate score ($-Log(p_{value})$)', color='g')

###############################################################################
# Compare to the weights of an SVM
clf = svm.SVC(kernel='linear')
clf.fit(X, y)

svm_weights = (clf.coef_ ** 2).sum(axis=0)
svm_weights /= svm_weights.max()

plt.bar(X_indices - .25, svm_weights, width=.2, label='SVM weight', color='r')

clf_selected = svm.SVC(kernel='linear')
clf_selected.fit(selector.transform(X), y)

svm_weights_selected = (clf_selected.coef_ ** 2).sum(axis=0)
svm_weights_selected /= svm_weights_selected.max()

plt.bar(X_indices[selector.get_support()] - .05, svm_weights_selected,
        width=.2, label='SVM weights after selection', color='b')


plt.title("Comparing feature selection")
plt.xlabel('Feature number')
plt.yticks(())
plt.axis('tight')
plt.legend(loc='upper right')
plt.show()
```

特徵選擇/範例七: Comparison of F-test and mutual information

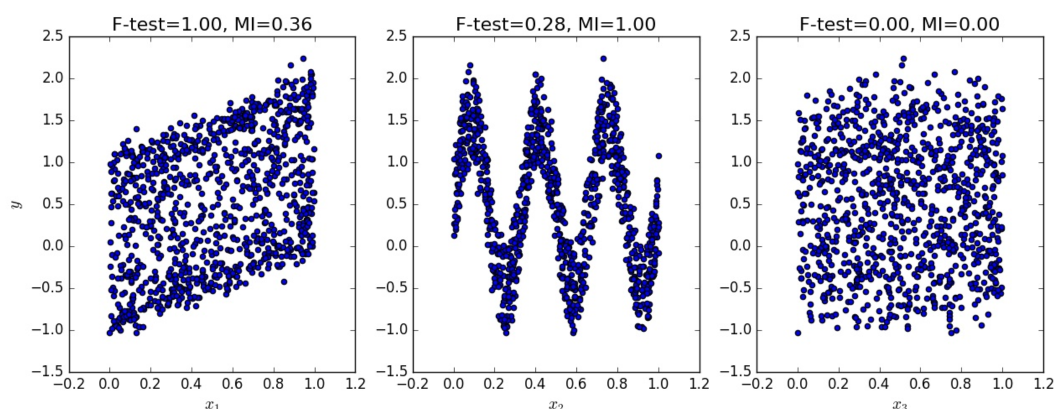
http://scikit-learn.org/stable/auto_examples/feature_selection/plot_f_test_vs_mi.html#sphx-glr-auto-examples-feature-selection-plot-f-test-vs-mi-py

這個範例是解釋單變量選擇特徵的兩個方法，F-test statistics以及mutual information。單變量特徵選擇可以算是選擇特徵的預處理，用以判斷適當的特徵選擇方式。

此範例假設了三個特徵變數x1, x2, x3分布在0與1之間，並且依照下列公式模擬預測目標： y = x1+ +sin(6 *pi* x2)+0.1 * N(0,1) 第三個特徵變量與預測目標無相關

下面的函式畫出了y與每個x_i之間的相依性，並且把F-test statistics以及mutual information的計算分數算出來，可以看到不同的變數影響方式在兩種方法會有不同的結果。

F-test 的結果只會關注線性相關的變數影響，該方法選擇x1作爲最具有特徵影響力的變量。另一方面，mutual information方法可以選出經過不同函式呈現的目標變數特徵，而他選擇了X2作爲最具有影響力的特徵，我們在直覺上認爲能找出經過三角函數轉換過的特徵變數，更符合在這個例子中目標變數的影響方式。而兩種方法都準確的判斷x3與目標變數無相關性。



```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_selection import f_regression, mutual_info_regression

np.random.seed(0)
X = np.random.rand(1000, 3)
y = X[:, 0] + np.sin(6 * np.pi * X[:, 1]) + 0.1 * np.random.randn(1000)

f_test, _ = f_regression(X, y)
f_test /= np.max(f_test)

mi = mutual_info_regression(X, y)
mi /= np.max(mi)

plt.figure(figsize=(15, 5))
for i in range(3):
    plt.subplot(1, 3, i + 1)
    plt.scatter(X[:, i], y)
    plt.xlabel("$x_{}$".format(i + 1), fontsize=14)
    if i == 0:
        plt.ylabel("$y$", fontsize=14)
    plt.title("F-test={:.2f}, MI={:.2f}".format(f_test[i], mi[i]),
              fontsize=16)
plt.show()
```

Feature Selection

互分解 / 範例一:Compare cross decomposition methods

http://scikit-learn.org/stable/auto_examples/cross_decomposition/plot_compare_cross_decomposition.html

這個範例目的是比較幾個互分解的方法。互分解運算主要是使用潛在變量模式（Latent variable modeling）分析來尋找兩個矩陣之間的主要相關成份。 對比於外顯變量（Manifest variable），也就是一般的觀察變量（Observational variable)，潛在變量是可能會影響實驗觀察的一個未知因素。

(一)引入函式庫及內建手寫數字資料庫

引入之函式庫如下

1. matplotlib.pyplot: 用來繪製影像
2. sklearn.cross_decomposition: 互分解物件
3. PLSCanonical: Partial Least Squares 淨最小平方法
4. PLSRegression: PLS 淨最小平方迴歸法
5. CCA: Canonical correlation analysis 典型相關分析

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cross_decomposition import PLSCanonical, PLSRegression, CCA

#首先產生500筆常態分佈資料
n = 500
# 共有兩個潛在變量:
l1 = np.random.normal(size=n)
l2 = np.random.normal(size=n)

# np.array([l1, l1, l2, l2]).shape = (4L, 500L)
# latents 為 500 x 4 之矩陣
latents = np.array([l1, l1, l2, l2]).T

#接下來加入亂數形成X, Y矩陣
X = latents + np.random.normal(size=4 * n).reshape((n, 4))
Y = latents + np.random.normal(size=4 * n).reshape((n, 4))

X_train = X[:n / 2]
Y_train = Y[:n / 2]
X_test = X[n / 2:]
Y_test = Y[n / 2:]

# numpy.corrcoef(x, y=None) 用來記算 Pearson product-moment 相關係數
print("Corr(X)")
print(np.round(np.corrcoef(X.T), 2))
print("Corr(Y)")
print(np.round(np.corrcoef(Y.T), 2))
```

```
Corr(X)
[[ 1.    0.48  0.02  0.  ]
 [ 0.48  1.    0.02 -0.02]
 [ 0.02  0.02  1.    0.51]
 [ 0.   -0.02  0.51  1.  ]]
Corr(Y)
[[ 1.    0.49 -0.01  0.05]
 [ 0.49  1.   -0.06  0.06]
 [-0.01 -0.06  1.    0.53]
 [ 0.05  0.06  0.53  1.  ]]
```

```
# Canonical (symmetric) PLS

# Transform data
# ~~~~~~~~~~~~~~
plsca = PLSCanonical(n_components=2)
plsca.fit(X_train, Y_train)
X_train_r, Y_train_r = plsca.transform(X_train, Y_train)
X_test_r, Y_test_r = plsca.transform(X_test, Y_test)

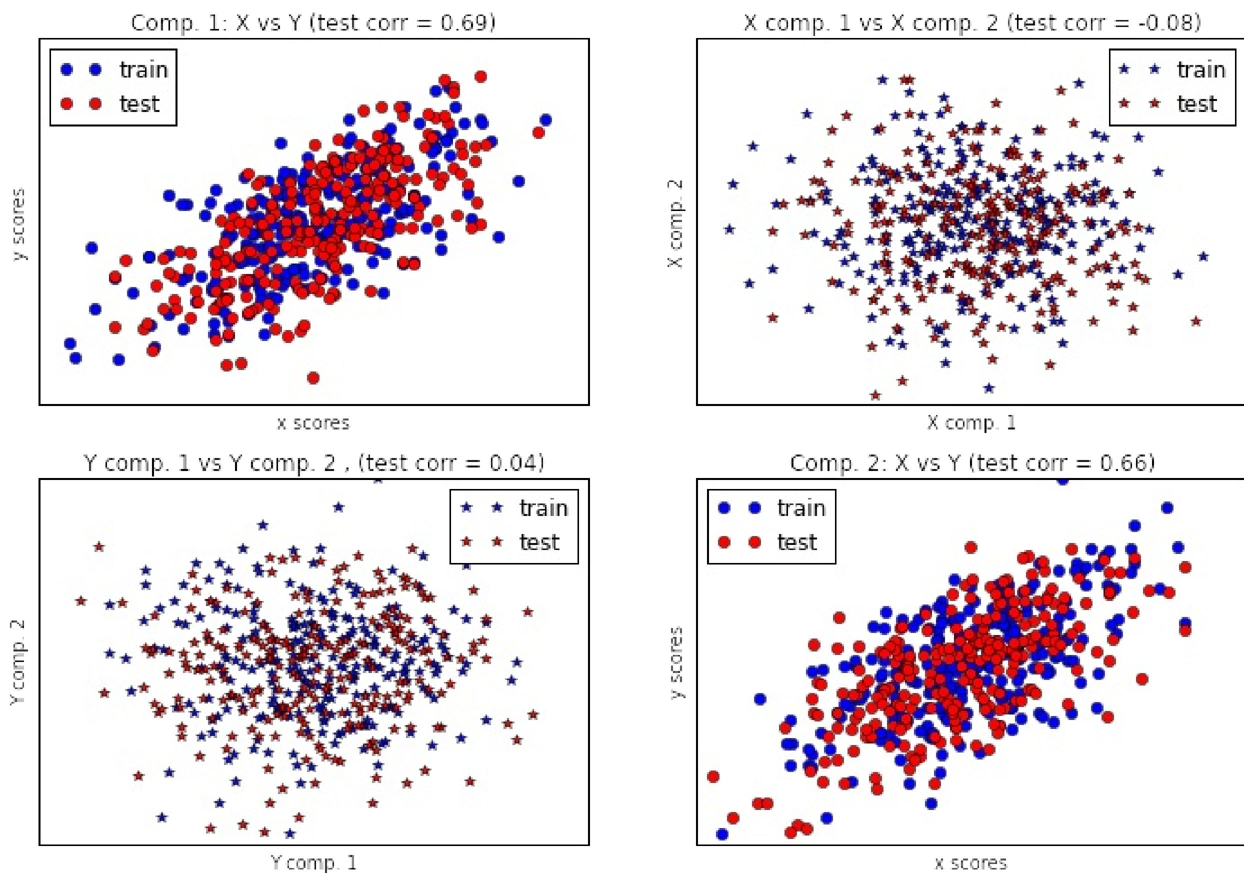
# Scatter plot of scores
# ~~~~~~~~~~~~~~~~~~~~~~
# 1) On diagonal plot X vs Y scores on each components
#figure = plt.figure(figsize=(30,20), dpi=300)
plt.figure(figsize=(12, 8), dpi=600)
plt.subplot(221)
plt.plot(X_train_r[:, 0], Y_train_r[:, 0], "ob", label="train")
plt.plot(X_test_r[:, 0], Y_test_r[:, 0], "or", label="test")
plt.xlabel("x scores")
plt.ylabel("y scores")
plt.title('Comp. 1: X vs Y (test corr = %.2f)' %
          np.corrcoef(X_test_r[:, 0], Y_test_r[:, 0])[0, 1])
plt.xticks(())
plt.yticks(())
plt.legend(loc="best")

plt.subplot(224)
plt.plot(X_train_r[:, 1], Y_train_r[:, 1], "ob", label="train")
plt.plot(X_test_r[:, 1], Y_test_r[:, 1], "or", label="test")
plt.xlabel("x scores")
plt.ylabel("y scores")
plt.title('Comp. 2: X vs Y (test corr = %.2f)' %
          np.corrcoef(X_test_r[:, 1], Y_test_r[:, 1])[0, 1])
plt.xticks(())
plt.yticks(())
plt.legend(loc="best")

# 2) Off diagonal plot components 1 vs 2 for X and Y
plt.subplot(222)
plt.plot(X_train_r[:, 0], X_train_r[:, 1], "*b", label="train")
plt.plot(X_test_r[:, 0], X_test_r[:, 1], "*r", label="test")
```

```
plt.xlabel("X comp. 1")
plt.ylabel("X comp. 2")
plt.title('X comp. 1 vs X comp. 2 (test corr = %.2f)'
          % np.corrcoef(X_test_r[:, 0], X_test_r[:, 1])[0, 1])
plt.legend(loc="best")
plt.xticks(())
plt.yticks(())

plt.subplot(223)
plt.plot(Y_train_r[:, 0], Y_train_r[:, 1], "*b", label="train")
plt.plot(Y_test_r[:, 0], Y_test_r[:, 1], "*r", label="test")
plt.xlabel("Y comp. 1")
plt.ylabel("Y comp. 2")
plt.title('Y comp. 1 vs Y comp. 2 , (test corr = %.2f)'
          % np.corrcoef(Y_test_r[:, 0], Y_test_r[:, 1])[0, 1])
plt.legend(loc="best")
plt.xticks(())
plt.yticks(())
plt.show()
```



```
###############################################################################
# PLS regression, with multivariate response, a.k.a. PLS2

n = 1000
q = 3
p = 10
X = np.random.normal(size=n * p).reshape((n, p))
B = np.array([[1, 2] + [0] * (p - 2)] * q).T
# each Yj = 1*X1 + 2*X2 + noize
Y = np.dot(X, B) + np.random.normal(size=n * q).reshape((n, q)) + 5

pls2 = PLSRegression(n_components=3)
pls2.fit(X, Y)
print("True B (such that: Y = XB + Err)")
print(B)
# compare pls2.coef_ with B
print("Estimated B")
print(np.round(pls2.coef_, 1))
pls2.predict(X)

###############################################################################
# PLS regression, with univariate response, a.k.a. PLS1

n = 1000
p = 10
X = np.random.normal(size=n * p).reshape((n, p))
y = X[:, 0] + 2 * X[:, 1] + np.random.normal(size=n * 1) + 5
pls1 = PLSRegression(n_components=3)
pls1.fit(X, y)
# note that the number of compements exceeds 1 (the dimension of y)
print("Estimated betas")
print(np.round(pls1.coef_, 1))

###############################################################################
# CCA (PLS mode B with symmetric deflation)

cca = CCA(n_components=2)
cca.fit(X_train, Y_train)
X_train_r, Y_train_r = plsca.transform(X_train, Y_train)
X_test_r, Y_test_r = plsca.transform(X_test, Y_test)
```

```
True B (such that: Y = XB + Err)
[[1 1 1]
 [2 2 2]
 [0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
Estimated B
[[ 1.   1.   1. ]
 [ 2.   1.9  2. ]
 [ 0.   0.   0. ]
 [ 0.   0.   0. ]
 [ 0.   0.   0. ]
 [ 0.   0.  -0.1]
 [ 0.   0.   0. ]
 [ 0.   0.   0.1]
 [ 0.   0.   0. ]
 [ 0.   0.   0. ]]
Estimated betas
[[ 1. ]
 [ 2. ]
 [ 0. ]
 [ 0. ]
 [ 0. ]
 [ 0. ]
 [ 0. ]
 [-0.1]
 [ 0. ]
 [ 0. ]]
```

(四)完整程式碼

Python source code: plot_compare_cross_decomposition.py

http://scikit-learn.org/stable/_downloads/plot_compare_cross_decomposition.py

```
print(__doc__)

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cross_decomposition import PLSCanonical, PLSRegression, CCA

###############################################################################
# Dataset based latent variables model

n = 500
# 2 latents vars:
l1 = np.random.normal(size=n)
```

```
l2 = np.random.normal(size=n)

latents = np.array([l1, l1, l2, l2]).T
X = latents + np.random.normal(size=4 * n).reshape((n, 4))
Y = latents + np.random.normal(size=4 * n).reshape((n, 4))

X_train = X[:n / 2]
Y_train = Y[:n / 2]
X_test = X[n / 2:]
Y_test = Y[n / 2:]

print("Corr(X)")
print(np.round(np.corrcoef(X.T), 2))
print("Corr(Y)")
print(np.round(np.corrcoef(Y.T), 2))

###############################################################################
# Canonical (symmetric) PLS

# Transform data
# ~~~~~~~~~~~~~~
plsca = PLSCanonical(n_components=2)
plsca.fit(X_train, Y_train)
X_train_r, Y_train_r = plsca.transform(X_train, Y_train)
X_test_r, Y_test_r = plsca.transform(X_test, Y_test)

# Scatter plot of scores
# ~~~~~~~~~~~~~~~~~~~~~~
# 1) On diagonal plot X vs Y scores on each components
plt.figure(figsize=(12, 8))
plt.subplot(221)
plt.plot(X_train_r[:, 0], Y_train_r[:, 0], "ob", label="train")
plt.plot(X_test_r[:, 0], Y_test_r[:, 0], "or", label="test")
plt.xlabel("x scores")
plt.ylabel("y scores")
plt.title('Comp. 1: X vs Y (test corr = %.2f)' %
          np.corrcoef(X_test_r[:, 0], Y_test_r[:, 0])[0, 1])
plt.xticks(())
plt.yticks(())
plt.legend(loc="best")

plt.subplot(224)
plt.plot(X_train_r[:, 1], Y_train_r[:, 1], "ob", label="train")
plt.plot(X_test_r[:, 1], Y_test_r[:, 1], "or", label="test")
plt.xlabel("x scores")
plt.ylabel("y scores")
plt.title('Comp. 2: X vs Y (test corr = %.2f)' %
          np.corrcoef(X_test_r[:, 1], Y_test_r[:, 1])[0, 1])
plt.xticks(())
plt.yticks(())
plt.legend(loc="best")

# 2) Off diagonal plot components 1 vs 2 for X and Y
```

```
plt.subplot(222)
plt.plot(X_train_r[:, 0], X_train_r[:, 1], "*b", label="train")
plt.plot(X_test_r[:, 0], X_test_r[:, 1], "*r", label="test")
plt.xlabel("X comp. 1")
plt.ylabel("X comp. 2")
plt.title('X comp. 1 vs X comp. 2 (test corr = %.2f)'
          % np.corrcoef(X_test_r[:, 0], X_test_r[:, 1])[0, 1])
plt.legend(loc="best")
plt.xticks(())
plt.yticks(())

plt.subplot(223)
plt.plot(Y_train_r[:, 0], Y_train_r[:, 1], "*b", label="train")
plt.plot(Y_test_r[:, 0], Y_test_r[:, 1], "*r", label="test")
plt.xlabel("Y comp. 1")
plt.ylabel("Y comp. 2")
plt.title('Y comp. 1 vs Y comp. 2 , (test corr = %.2f)'
          % np.corrcoef(Y_test_r[:, 0], Y_test_r[:, 1])[0, 1])
plt.legend(loc="best")
plt.xticks(())
plt.yticks(())
plt.show()

###############################################################################
# PLS regression, with multivariate response, a.k.a. PLS2

n = 1000
q = 3
p = 10
X = np.random.normal(size=n * p).reshape((n, p))
B = np.array([[1, 2] + [0] * (p - 2)] * q).T
# each Yj = 1*X1 + 2*X2 + noize
Y = np.dot(X, B) + np.random.normal(size=n * q).reshape((n, q)) + 5

pls2 = PLSRegression(n_components=3)
pls2.fit(X, Y)
print("True B (such that: Y = XB + Err)")
print(B)
# compare pls2.coef_ with B
print("Estimated B")
print(np.round(pls2.coef_, 1))
pls2.predict(X)

###############################################################################
# PLS regression, with univariate response, a.k.a. PLS1

n = 1000
p = 10
X = np.random.normal(size=n * p).reshape((n, p))
y = X[:, 0] + 2 * X[:, 1] + np.random.normal(size=n * 1) + 5
pls1 = PLSRegression(n_components=3)
pls1.fit(X, y)
# note that the number of compements exceeds 1 (the dimension of y)
```

```
print("Estimated betas")
print(np.round(pls1.coef_, 1))

###############################################################################
# CCA (PLS mode B with symmetric deflation)

cca = CCA(n_components=2)
cca.fit(X_train, Y_train)
X_train_r, Y_train_r = plsca.transform(X_train, Y_train)
X_test_r, Y_test_r = plsca.transform(X_test, Y_test)
```

通用範例 General Examples

通用範例/範例一: Plotting Cross-Validated Predictions

http://scikit-learn.org/stable/auto_examples/plot_cv_predict.html

1. 資料集：波士頓房產
2. 特徵：房地產客觀數據，如年份、平面大小
3. 預測目標：房地產價格
4. 機器學習方法：線性迴歸
5. 探討重點：10 等分的交叉驗証(10-fold Cross-Validation)來實際測試資料以及預測值的關係
6. 關鍵函式： `sklearn.cross_validation.cross_val_predict`

(一)引入函式庫及內建測試資料庫

引入之函式庫如下

1. `matplotlib.pyplot` : 用來繪製影像
2. `sklearn.datasets` : 用來繪入內建測試資料庫
3. `sklearn.cross_validation import cross_val_predict` : 利用交叉驗證的方式來預測
4. `sklearn.linear_model` : 使用線性迴歸

(二)引入內建測試資料庫(boston房產資料)

使用 `datasets.load_boston()` 將資料存入， `boston` 為一個dict型別資料，我們看一下資料的內容。

```
lr = linear_model.LinearRegression()
#lr = LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
boston = datasets.load_boston()
y = boston.target
```

顯示	說明
('data', (506, 13))	房地產的資料集，共506筆房產13個特徵
('feature_names', (13,))	房地產的特徵名
('target', (506,))	回歸目標
DESCR	資料之描述

(三) `cross_val_predict` 的使用

`sklearn.cross_validation.cross_val_predict` (estimator, X, y=None, cv=None, n_jobs=1, verbose=0, fit_params=None, pre_dispatch='2*n_jobs')

X為機器學習數據， y為回歸目標， cv為交叉驗証時資料切分的依據，範例為10則將資料切分為10等分，以其中9等分為訓練集，另外一等分則為測試集。

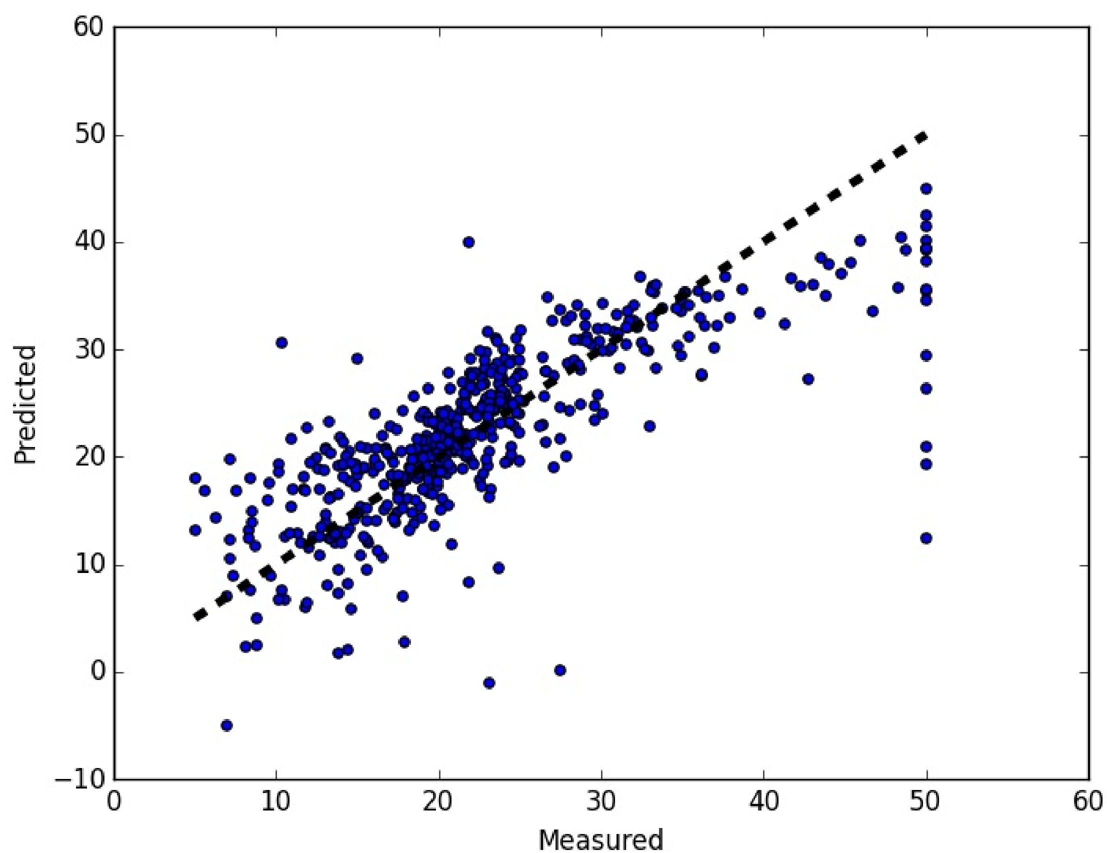
```
predicted = cross_val_predict(lr, boston.data, y, cv=10)
```

(四)繪出預測結果與實際目標差異圖

X軸為回歸目標，Y軸為預測結果。

並劃出一條斜率=1的理想曲線(用虛線標示)

```
fig, ax = plt.subplots()
ax.scatter(y, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
plt.show()
```



(五)完整程式碼

Python source code: plot_cv_predict.py

http://scikit-learn.org/stable/auto_examples/plot_cv_predict.html

```
from sklearn import datasets
from sklearn.cross_validation import cross_val_predict
from sklearn import linear_model
import matplotlib.pyplot as plt

lr = linear_model.LinearRegression()
boston = datasets.load_boston()
y = boston.target

# cross_val_predict returns an array of the same size as `y` where each entry
# is a prediction obtained by cross validated:
predicted = cross_val_predict(lr, boston.data, y, cv=10)

fig, ax = plt.subplots()
ax.scatter(y, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
plt.show()
```

通用範例/範例二: Concatenating multiple feature extraction methods

http://scikit-learn.org/stable/auto_examples/feature_stacker.html

在許多實際應用中，會有很多方法可以從一個數據集中提取特徵。也常常會組合多個方法來獲得良好的特徵。這個例子說明如何使用 `FeatureUnion` 來結合由 `PCA` 和 `univariate selection` 時的特徵。

這個範例的主要目的：

1. 資料集：iris 鳶尾花資料集
2. 特徵：鳶尾花特徵
3. 預測目標：是那一種鳶尾花
4. 機器學習方法：SVM 支持向量機
5. 探討重點：特徵結合
6. 關鍵函式： `sklearn.pipeline.FeatureUnion`

(一)資料匯入及描述

- 首先先匯入iris 鳶尾花資料集，使用from sklearn.datasets import load_iris將資料存入
- 準備X (特徵資料) 以及 y (目標資料)

```
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.grid_search import GridSearchCV
from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest

iris = load_iris()

X, y = iris.data, iris.target
```

測試資料：
`iris` 為一個dict型別資料。

顯示	說明
('target_names', (3L,))	共有三種鳶尾花 setosa, versicolor, virginica
('data', (150L, 4L))	有150筆資料，共四種特徵
('target', (150L,))	這150筆資料各是那一種鳶尾花
DESCR	資料之描述
feature_names	4個特徵代表的意義

(二)PCA與SelectKBest

- `PCA(n_components = 主要成份數量)` :Principal Component Analysis(PCA)主成份分析，是一個常用的將資料維度減少的方法。它的原理是找出一個新的座標軸，將資料投影到該軸時，數據的變異量會最大。利用這個方式減少資料維度，又希望能保留住原數據點的特性。

- `SelectKBest(score_func , k )` : `score_func` 是選擇特徵值所依據的函式，而 `K` 值則是設定要選出多少特徵。

```
# This dataset is way to high-dimensional. Better do PCA:
pca = PCA(n_components=2)

# Maybe some original features where good, too?
selection = SelectKBest(k=1)
```

(三)FeatureUnionc

- 使用sklearn.pipeline.FeatureUnion合併主成分分析(PCA)和綜合篩選(SelectKBest)。
- 最後得到選出的特徵

```
# Build estimator from PCA and Univariate selection:

combined_features = FeatureUnion([("pca", pca), ("univ_select", selection)])

# Use combined features to transform dataset:
X_features = combined_features.fit(X, y).transform(X)
```

(四)找到最佳的結果

- Scikit-learn的支持向量機分類函式庫利用 SVC() 建立運算物件，之後並可以用運算物件內的方法 .fit() 與 .predict() 來做訓練與預測。
- 使用 `GridSearchCV` 交叉驗證，得到由參數網格計算出的分數網格，並找到分數網格中最佳點。最後顯示這個點所代表的參數

```
svm = SVC(kernel="linear")

# Do grid search over k, n_components and C:

pipeline = Pipeline([("features", combined_features), ("svm", svm)])

param_grid = dict(features__pca__n_components=[1, 2, 3],
                  features__univ_select__k=[1, 2],
                  svm__C=[0.1, 1, 10])

grid_search = GridSearchCV(pipeline, param_grid=param_grid, verbose=10)
grid_search.fit(X, y)
print(grid_search.best_estimator_)
```

結果顯示 ``` Fitting 3 folds for each of 18 candidates, totalling 54 fits [CV] features**univ_select**k=1, features**pca**n_components=1, svm**C=0.1 [CV] features**univ_select**k=1, features**pca**n_components=1, svm**C=0.1, score=0.960784 - 0.0s

````
## (五)完整程式碼
Python source code: feature_stacker.py
http://scikit-learn.org/stable/auto_examples/feature_stacker.html

```python
# Author: Andreas Mueller <amueller@ais.uni-bonn.de>
#
# License: BSD 3 clause

from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.grid_search import GridSearchCV
from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest

iris = load_iris()

X, y = iris.data, iris.target

# This dataset is way to high-dimensional. Better do PCA:
pca = PCA(n_components=2)

# Maybe some original features where good, too?
selection = SelectKBest(k=1)

# Build estimator from PCA and Univariate selection:

combined_features = FeatureUnion([("pca", pca), ("univ_select", selection)])

# Use combined features to transform dataset:
X_features = combined_features.fit(X, y).transform(X)

svm = SVC(kernel="linear")

# Do grid search over k, n_components and C:

pipeline = Pipeline([("features", combined_features), ("svm", svm)])

param_grid = dict(features__pca__n_components=[1, 2, 3],
                  features__univ_select__k=[1, 2],
                  svm__C=[0.1, 1, 10])

grid_search = GridSearchCV(pipeline, param_grid=param_grid, verbose=10)
grid_search.fit(X, y)
print(grid_search.best_estimator_)
````
````

通用範例/範例三: Isotonic Regression

http://scikit-learn.org/stable/auto_examples/plot_isotonic_regression.html

迴歸函數採用遞增函數。

- y[] are inputs (real numbers)
- y_[] are fitted

這個範例的主要目的：

比較

- Isotonic Fit
- Linear Fit

(一) Regression「迴歸」

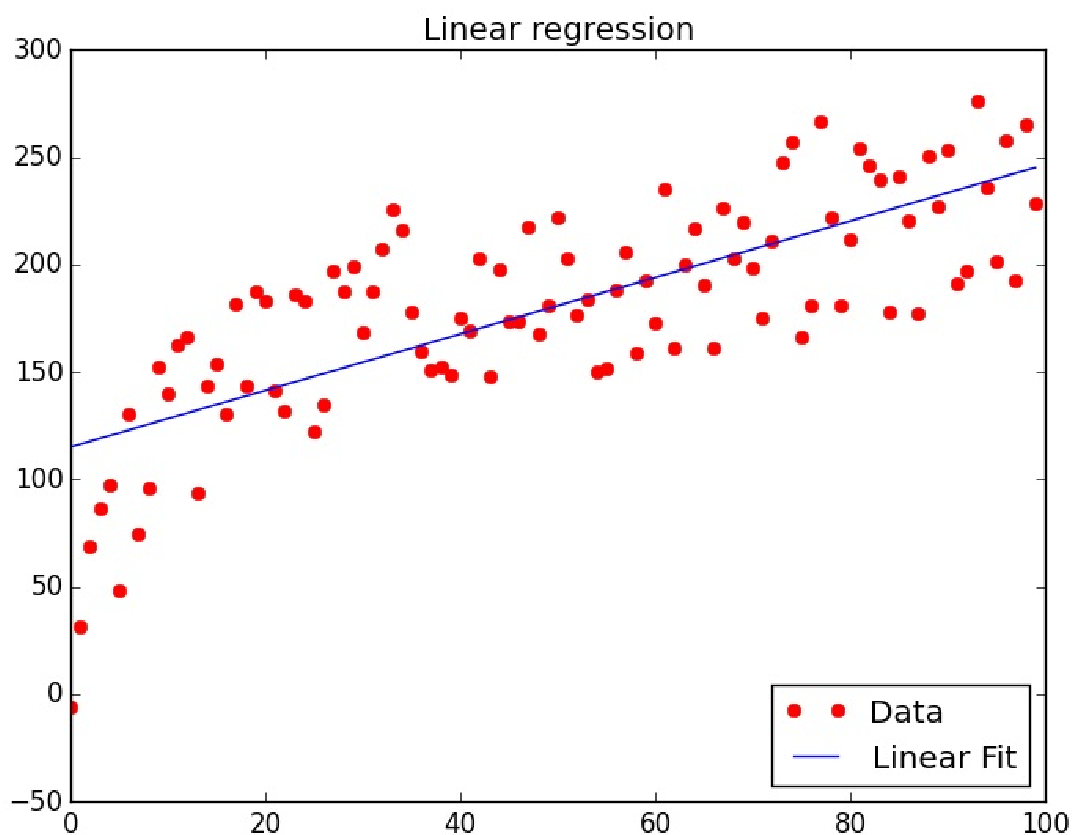
「迴歸」就是找一個函數，盡量符合手邊的一堆數據。此函數稱作「迴歸函數」。

(二) Linear Regression「線性迴歸」

迴歸函數採用線性函數。誤差採用平方誤差。

`class sklearn.linear_model.LinearRegression`

二維數據，迴歸函數是直線。

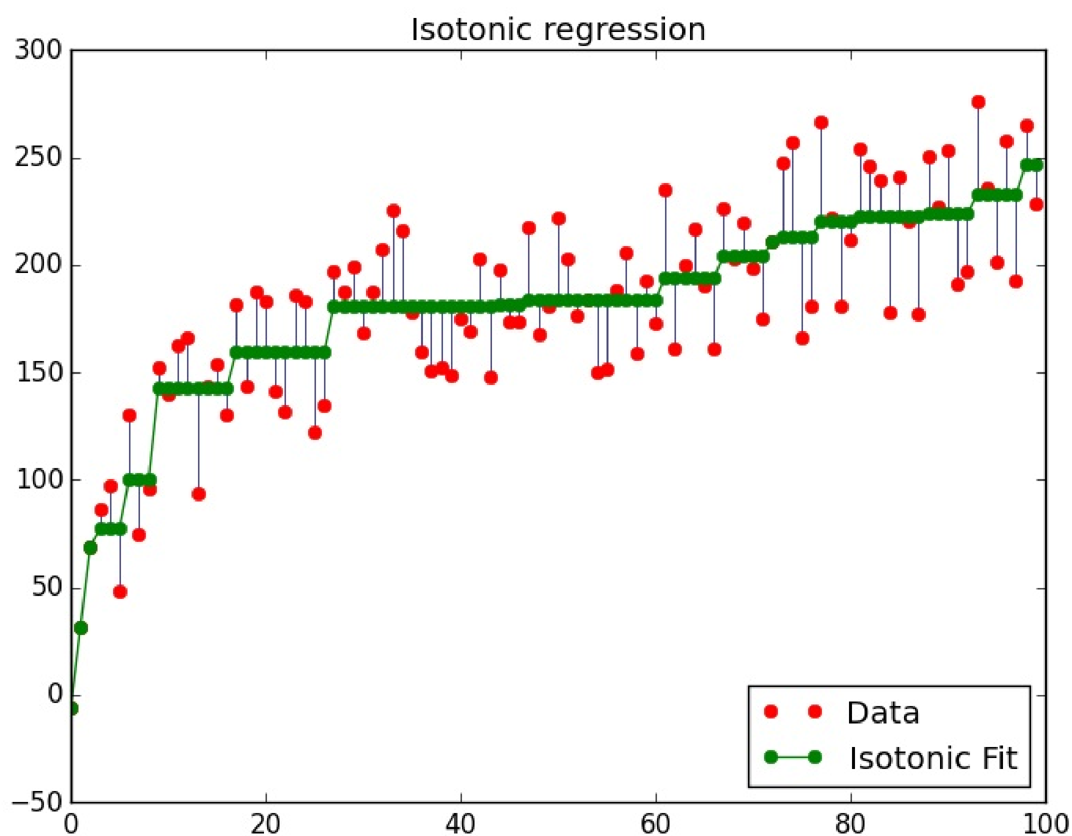


(三) Isotonic Regression「保序迴歸」

具有分段迴歸的效果。迴歸函數採用遞增函數。

`class sklearn.isotonic.IsotonicRegression`

採用平方誤差，時間複雜度 O(N) 。



(四) 完整程式碼

Python source code: plot_isotonic_regression.py

http://scikit-learn.org/stable/auto_examples/plot_isotonic_regression.html

```
print(__doc__)

# Author: Nelle Varoquaux <nelle.varoquaux@gmail.com>
#         Alexandre Gramfort <alexandre.gramfort@inria.fr>
# Licence: BSD

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.collections import LineCollection

from sklearn.linear_model import LinearRegression
from sklearn.isotonic import IsotonicRegression
from sklearn.utils import check_random_state

n = 100
x = np.arange(n)
rs = check_random_state(0)
y = rs.randint(-50, 50, size=(n,)) + 50. * np.log(1 + np.arange(n))

###############################################################################
# Fit IsotonicRegression and LinearRegression models

ir = IsotonicRegression()

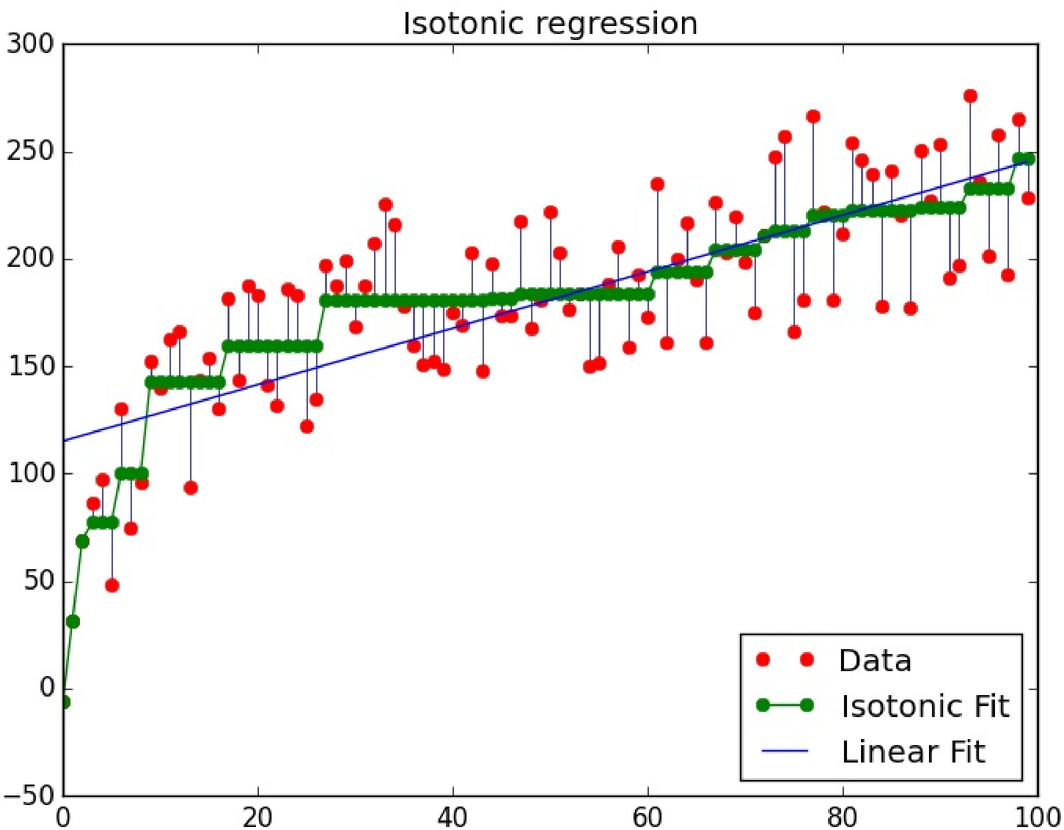
y_ = ir.fit_transform(x, y)

lr = LinearRegression()
lr.fit(x[:, np.newaxis], y)  # x needs to be 2d for LinearRegression

###############################################################################
# plot result

segments = [[[i, y[i]], [i, y_[i]]] for i in range(n)]
lc = LineCollection(segments, zorder=0)
lc.set_array(np.ones(len(y)))
lc.set_linewidths(0.5 * np.ones(n))

fig = plt.figure()
plt.plot(x, y, 'r.', markersize=12)
plt.plot(x, y_, 'g.-', markersize=12)
plt.plot(x, lr.predict(x[:, np.newaxis]), 'b-')
plt.gca().add_collection(lc)
plt.legend(('Data', 'Isotonic Fit', 'Linear Fit'), loc='lower right')
plt.title('Isotonic regression')
plt.show()
```


Isotonic regression
300
250
200
150
100
50
0
−50
0
20
40
60
80
100
Data
Isotonic Fit
Linear Fit

通用範例/範例四: Imputing missing values before building an estimator

http://scikit-learn.org/stable/auto_examples/missing_values.htm

在這範例說明有時補充缺少的數據(missing values)，可以得到更好的結果。但仍然需要進行交叉驗證。來驗證填充是否合適。而missing values可以用均值、中位值，或者頻繁出現的值代替。中位值對大數據之機器學習來說是比較穩定的估計值。

(一)引入函式庫及內建測試資料庫

引入之函式庫如下

1. `sklearn.ensemble.RandomForestRegressor` : 隨機森林回歸
2. `sklearn.pipeline.Pipeline` : 串聯估計器
3. `sklearn.preprocessing.Imputer` : 缺失值填充
4. `sklearn.cross_validation import cross_val_score` :交叉驗證

(二)引入內建測試資料庫(boston房產資料)

使用 `datasets.load_boston()` 將資料存入， `boston` 為一個dict型別資料，我們看一下資料的內容。
n_samples 為樣本數
n_features 為特徵數

```
dataset = load_boston()
X_full, y_full = dataset.data, dataset.target
n_samples = X_full.shape[0]
n_features = X_full.shape[1]
```

顯示	說明
('data', (506, 13))	機器學習數據
('feature_names', (13,))	房地產相關特徵
('target', (506,))	回歸目標
DESCR	資料之描述

共有506筆資料及13個特徵('CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD','TAX', 'PTRATIO', 'B', 'LSTAT')用來描述房地產的週邊狀況，如CRIM (per capita crime rate by town)跟該區域之犯罪率有關。而迴歸目標為房地產的價格，以1000美元為單位。也就是說這個範例希望以房地產的週遭客觀數據來預測房地產的價格。

(三)利用整個數據集來預測

全部的資料使用隨機森林回歸函數進行交叉驗證，得到一個分數。

Score with the entire dataset = 0.56

```
estimator = RandomForestRegressor(random_state=0, n_estimators=100)
score = cross_val_score(estimator, X_full, y_full).mean()
print("Score with the entire dataset = %.2f" % score)
```

(四)模擬資料損失時之預測情形

設定損失比例，並估計移除missing values後的得分 損失比例75%，損失樣本數為379筆，剩餘樣本為127筆。
將127筆資料進行隨機森林回歸函數進行交叉驗證，並得到一個分數。

Score without the samples containing missing values = 0.49

```
missing_rate = 0.75
n_missing_samples = np.floor(n_samples * missing_rate)
missing_samples = np.hstack((np.zeros(n_samples - n_missing_samples,
                                      dtype=np.bool),
                             np.ones(n_missing_samples,
                                     dtype=np.bool)))
rng.shuffle(missing_samples)
missing_features = rng.randint(0, n_features, n_missing_samples)

X_filtered = X_full[~missing_samples, :]
y_filtered = y_full[~missing_samples]
estimator = RandomForestRegressor(random_state=0, n_estimators=100)
score = cross_val_score(estimator, X_filtered, y_filtered).mean()
print("Score without the samples containing missing values = %.2f" % score)
```

(五)填充**missing values**，估計填充後的得分

每一筆樣本資料都在13個特徵中隨機遺失一個特徵資料，
使用 `sklearn.preprocessing.Imputer` 進行missing values的填充。

```
class sklearn.preprocessing.Imputer(missing_values='NaN', strategy='mean', axis=0, verbose=0, c
opy=True)
```

填充後進行隨機森林回歸函數進行交叉驗證，獲得填充後分數。

```
X_missing = X_full.copy()
X_missing[np.where(missing_samples)[0], missing_features] = 0
y_missing = y_full.copy()
estimator = Pipeline([("imputer", Imputer(missing_values=0,
                                          strategy="mean",
                                          axis=0)),
                      ("forest", RandomForestRegressor(random_state=0,
                                                       n_estimators=100))])
score = cross_val_score(estimator, X_missing, y_missing).mean()
print("Score after imputation of the missing values = %.2f" % score)
```

利用數據填充後的迴歸函數，去測試填充前的資料，預測的準確率獲得提升。

Score after imputation of the missing values = 0.57

(六)完整程式碼

Python source code: missing_values.py
http://scikit-learn.org/stable/auto_examples/missing_values.html#example-missing-values-py

```
import numpy as np

from sklearn.datasets import load_boston
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Imputer
from sklearn.cross_validation import cross_val_score

rng = np.random.RandomState(0)

dataset = load_boston()
X_full, y_full = dataset.data, dataset.target
n_samples = X_full.shape[0]
n_features = X_full.shape[1]

# Estimate the score on the entire dataset, with no missing values
estimator = RandomForestRegressor(random_state=0, n_estimators=100)
score = cross_val_score(estimator, X_full, y_full).mean()
print("Score with the entire dataset = %.2f" % score)

# Add missing values in 75% of the lines
missing_rate = 0.75
n_missing_samples = np.floor(n_samples * missing_rate)
missing_samples = np.hstack((np.zeros(n_samples - n_missing_samples,
                                      dtype=np.bool),
                             np.ones(n_missing_samples,
                                     dtype=np.bool)))
rng.shuffle(missing_samples)
missing_features = rng.randint(0, n_features, n_missing_samples)

# Estimate the score without the lines containing missing values
X_filtered = X_full[~missing_samples, :]
y_filtered = y_full[~missing_samples]
estimator = RandomForestRegressor(random_state=0, n_estimators=100)
score = cross_val_score(estimator, X_filtered, y_filtered).mean()
print("Score without the samples containing missing values = %.2f" % score)

# Estimate the score after imputation of the missing values
X_missing = X_full.copy()
X_missing[np.where(missing_samples)[0], missing_features] = 0
y_missing = y_full.copy()
estimator = Pipeline([("imputer", Imputer(missing_values=0,
                                          strategy="mean",
                                          axis=0)),
                      ("forest", RandomForestRegressor(random_state=0,
                                                       n_estimators=100))])
score = cross_val_score(estimator, X_missing, y_missing).mean()
print("Score after imputation of the missing values = %.2f" % score)
```

```
results:
Score with the entire dataset = 0.56
Score without the samples containing missing values = 0.48
Score after imputation of the missing values = 0.55
```

通用範例/範例七: Face completion with a multi-output estimators

http://scikit-learn.org/stable/auto_examples/plot_multioutput_face_completion.html

這個範例用來展示scikit-learn如何用 `extremely randomized trees` , `k nearest neighbors` , `linear regression` 和 `ridge regression` 演算法來完成人臉估測。

(一)引入函式庫及內建影像資料庫

引入之函式庫如下

1. `sklearn.datasets` : 用來繪入內建之影像資料庫
2. `sklearn.utils.validation` : 用來取亂數
3. `sklearn.ensemble`
4. `sklearn.neighbors`
5. `sklearn.linear_model`

使用 `datasets.load_digits()` 將資料存入， `data` 為一個dict型別資料，我們看一下資料的內容。

```
from sklearn.datasets import fetch_olivetti_faces
data = fetch_olivetti_faces()
targets = data.target
data = data.images.reshape((len(data.images), -1))
```

顯示	說明
('images', (400, 64, 64))	共有40個人，每個人各有10張影像，共有 400 張影像，影像大小為 64x64
('data', (400, 4096))	data 則是將64x64的矩陣攤平成4096個元素之一維向量
('targets', (400,))	說明400張圖與40個人之分類對應 0-39，記錄每張影像是哪一個人
DESCR	資料之描述

前面30個人當訓練資料，之後當測試資料

```
train = data[targets < 30]
test = data[targets >= 30]
```

測試影像從100張亂數選5張出來，變數 `test` 的大小變成(5,4096)

```
# Test on a subset of people
n_faces = 5
rng = check_random_state(4)
face_ids = rng.randint(test.shape[0], size=(n_faces, ))
test = test[face_ids, :]
```

把每張訓練影像和測試影像都切割成上下兩部分:

X人臉上半部分， Y人臉下半部分。

```
n_pixels = data.shape[1]
X_train = train[:, :np.ceil(0.5 * n_pixels)]
y_train = train[:, np.floor(0.5 * n_pixels):]
X_test = test[:, :np.ceil(0.5 * n_pixels)]
y_test = test[:, np.floor(0.5 * n_pixels):]
```

(二)資料訓練

分別用以下四種演算法來完成人臉下半部估測

1. `extremely randomized trees` (絕對隨機森林演算法)
2. `k nearest neighbors` (K-鄰近演算法)
3. `linear regression` (線性回歸演算法)
4. `ridge regression` (脊回歸演算法)

```
ESTIMATORS = {
    "Extra trees": ExtraTreesRegressor(n_estimators=10, max_features=32,random_state=0),
    "K-nn": KNeighborsRegressor(),
    "Linear regression": LinearRegression(),
    "Ridge": RidgeCV(),
}
```

分別把訓練資料人臉上、下部分放入 `estimator.fit()` 中進行訓練。上半部分人臉爲條件影像，下半部人臉爲目標影像。

`y_test_predict` 爲一個dict型別資料，存放5位測試者分別用四種演算法得到的人臉下半部估計結果。

```
y_test_predict = dict()
for name, estimator in ESTIMATORS.items():
    estimator.fit(X_train, y_train)
    y_test_predict[name] = estimator.predict(X_test)
```

(三) `matplotlib.pyplot` 畫出結果

每張影像都是64*64，總共有5位測試者，每位測試者分別有1張原圖，加上使用4種演算法得到的估測結果。

```
image_shape = (64, 64)
n_cols = 1 + len(ESTIMATORS)
plt.figure(figsize=(2. * n_cols, 2.26 * n_faces))
plt.suptitle("Face completion with multi-output estimators", size=16)

for i in range(n_faces):
    true_face = np.hstack((X_test[i], y_test[i]))

    if i:
        sub = plt.subplot(n_faces, n_cols, i * n_cols + 1)
    else:
        sub = plt.subplot(n_faces, n_cols, i * n_cols + 1,
                          title="true faces")


    sub.axis("off")
    sub.imshow(true_face.reshape(image_shape),
               cmap=plt.cm.gray,
               interpolation="nearest")

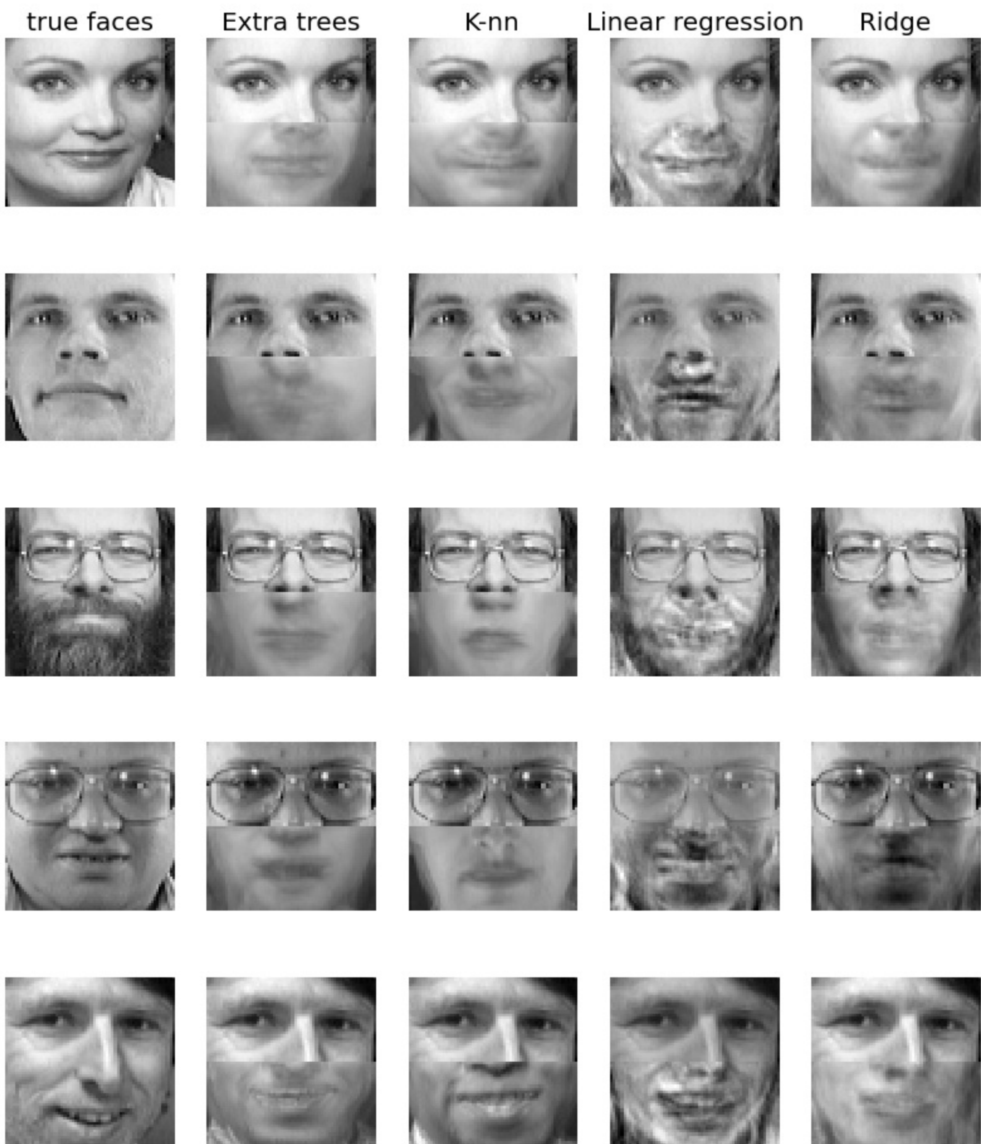
    for j, est in enumerate(sorted(ESTIMATORS)):
        completed_face = np.hstack((X_test[i], y_test_predict[est][i]))

        if i:
            sub = plt.subplot(n_faces, n_cols, i * n_cols + 2 + j)

        else:
            sub = plt.subplot(n_faces, n_cols, i * n_cols + 2 + j,
                              title=est)

        sub.axis("off")
        sub.imshow(completed_face.reshape(image_shape),
                   cmap=plt.cm.gray,
                   interpolation="nearest")

plt.show()
```


Face completion with multi-output estimators
true faces
Extra trees
K-nn
Linear regression
Ridge

群聚法 Clustering

範例十二:Spectral clustering for image segmentation

http://scikit-learn.org/stable/auto_examples/cluster/plot_segmentation_toy.html

此範例是利用Spectral clustering來區別重疊的圓圈，將重疊的圓圈分為個體。

1. 建立一個100x100的影像包含四個不同半徑的圓
2. 透過 `np.indices` 改變影像顏色複雜度
3. 用 `spectral_clustering` 區分出各個不同區域特徵

(一)引入函式庫

引入函式庫如下：

1. `numpy` :產生陣列數值
2. `matplotlib.pyplot` :用來繪製影像
3. `sklearn.feature_extraction import image` :將每個像素的梯度關係圖像化
4. `sklearn.cluster import spectral_clustering` :將影像正規化切割

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.feature_extraction import image
from sklearn.cluster import spectral_clustering
```

(二)建立要被區分的重疊圓圈影像

- 產生一個大小為輸入值得矩陣(此範例為100x100)，其內部值為沿著座標方向遞增(如:0,1,...)的值。

```
l = 100
x, y = np.indices((l, l))
```

- 建立四個圓圈的圓心座標並給定座標值
- 給定四個圓圈的半徑長度
- 將圓心座標與半徑結合產生四個圓圈圖像

```
center1 = (28, 24)
center2 = (40, 50)
center3 = (67, 58)
center4 = (24, 70)

radius1, radius2, radius3, radius4 = 16, 14, 15, 14

circle1 = (x - center1[0]) ** 2 + (y - center1[1]) ** 2 < radius1 ** 2
circle2 = (x - center2[0]) ** 2 + (y - center2[1]) ** 2 < radius2 ** 2
circle3 = (x - center3[0]) ** 2 + (y - center3[1]) ** 2 < radius3 ** 2
circle4 = (x - center4[0]) ** 2 + (y - center4[1]) ** 2 < radius4 ** 2
```

- 將上一段產生的四個圓圈影像合併為 `img` 使其成為一體的物件
- `mask` 為布林形式的 `img`

- `img` 為浮點數形式的 `img`
- 用亂數產生的方法將整張影像作亂數處理

```
# 4 circles
img = circle1 + circle2 + circle3 + circle4
mask = img.astype(bool)
img = img.astype(float)

img += 1 + 0.2 * np.random.randn(*img.shape)
```

接著將產生好的影像化為可使用 `spectral_clustering` 的影像

- `image.img_to_graph` 用來處理邊緣的權重與每個像速間的梯度關聯有關
- 用類似Voronoi Diagram演算法的概念來處理影像

```
graph = image.img_to_graph(img, mask=mask)

graph.data = np.exp(-graph.data / graph.data.std())
```

最後用 `spectral_clustering` 將連在一起的部分切開，而 `spectral_clustering` 中的各項參數設定如下:

- `graph` : 必須是一個矩陣且大小為nxn的形式
- `n_clusters=4` : 需要提取出的群集數
- `eigen_solver='arpack'` : 解特徵值的方式

開一張新影像 `label_im` 用來展示 `spectral_clustering` 切開後的分類結果

```
labels = spectral_clustering(graph, n_clusters=4, eigen_solver='arpack')
label_im = -np.ones(mask.shape)
label_im[mask] = labels

plt.matshow(img)
plt.matshow(label_im)
```



(三)完整程式碼

Python source code:plot_segmentation_toy.py

http://scikit-learn.org/stable/_downloads/plot_segmentation_toy.py

```
print(__doc__)

# Authors:  Emmanuelle Gouillart <emmanuelle.gouillart@normalesup.org>
#           Gael Varoquaux <gael.varoquaux@normalesup.org>
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt

from sklearn.feature_extraction import image
from sklearn.cluster import spectral_clustering
```

```
###############################################################################
l = 100
x, y = np.indices((l, l))

center1 = (28, 24)
center2 = (40, 50)
center3 = (67, 58)
center4 = (24, 70)

radius1, radius2, radius3, radius4 = 16, 14, 15, 14

circle1 = (x - center1[0]) ** 2 + (y - center1[1]) ** 2 < radius1 ** 2
circle2 = (x - center2[0]) ** 2 + (y - center2[1]) ** 2 < radius2 ** 2
circle3 = (x - center3[0]) ** 2 + (y - center3[1]) ** 2 < radius3 ** 2
circle4 = (x - center4[0]) ** 2 + (y - center4[1]) ** 2 < radius4 ** 2

###############################################################################
# 4 circles
img = circle1 + circle2 + circle3 + circle4
mask = img.astype(bool)
img = img.astype(float)

img += 1 + 0.2 * np.random.randn(*img.shape)

# Convert the image into a graph with the value of the gradient on the
# edges.
graph = image.img_to_graph(img, mask=mask)

# Take a decreasing function of the gradient: we take it weakly
# dependent from the gradient the segmentation is close to a voronoi
graph.data = np.exp(-graph.data / graph.data.std())

# Force the solver to be arpack, since amg is numerically
# unstable on this example
labels = spectral_clustering(graph, n_clusters=4, eigen_solver='arpack')
label_im = -np.ones(mask.shape)
label_im[mask] = labels

plt.matshow(img)
plt.matshow(label_im)

###############################################################################
# 2 circles
img = circle1 + circle2
mask = img.astype(bool)
img = img.astype(float)

img += 1 + 0.2 * np.random.randn(*img.shape)

graph = image.img_to_graph(img, mask=mask)
graph.data = np.exp(-graph.data / graph.data.std())
```

```
labels = spectral_clustering(graph, n_clusters=2, eigen_solver='arpack')
label_im = -np.ones(mask.shape)
label_im[mask] = labels

plt.matshow(img)
plt.matshow(label_im)

plt.show()
```

機器學習資料集 Datasets

這個章節介紹scikit-learn 所提供之機器學習資料集，最常用的主要有:

- 手寫數字辨識
- 鳶尾花資料集

Datasets

機器學習資料集/ 範例一: The digits dataset

http://scikit-learn.org/stable/auto_examples/datasets/plot_digits_last_image.html

這個範例目的是介紹機器學習範例資料集的操作，對於初學者以及授課特別適合使用。

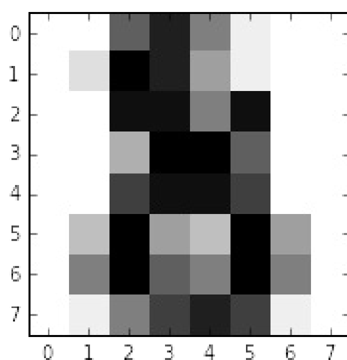
(一)引入函式庫及內建手寫數字資料庫

```
#這行是在ipython notebook的介面裏專用，如果在其他介面則可以拿掉
%matplotlib inline
from sklearn import datasets

import matplotlib.pyplot as plt

#載入數字資料集
digits = datasets.load_digits()

#畫出第一個圖片
plt.figure(1, figsize=(3, 3))
plt.imshow(digits.images[-1], cmap=plt.cm.gray_r, interpolation='nearest')
plt.show()
```



(二)資料集介紹

`digits = datasets.load_digits()` 將一個dict型別資料存入digits，我們可以用下面程式碼來觀察裏面資料

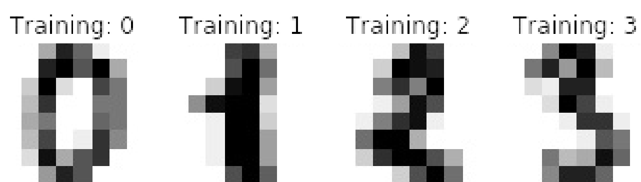
```
for key,value in digits.items() :
    try:
        print (key,value.shape)
    except:
        print (key)
```

```
('images', (1797L, 8L, 8L))
('data', (1797L, 64L))
('target_names', (10L,))
DESCR
('target', (1797L,))
```

顯示	說明
('images', (1797L, 8L, 8L))	共有 1797 張影像，影像大小爲 8x8
('data', (1797L, 64L))	data 則是將8x8的矩陣攤平成64個元素之一維向量
('target_names', (10L,))	說明10種分類之對應 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
DESCR	資料之描述
('target', (1797L,))	記錄1797張影像各自代表那一個數字

接下來我們試著以下面指令來觀察資料檔，每張影像所對照的實際數字存在 `digits.target` 變數中

```
images_and_labels = list(zip(digits.images, digits.target))
for index, (image, label) in enumerate(images_and_labels[:4]):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % label)
```




```
#接著我們嘗試將這個機器學習資料之描述檔顯示出來
print(digits['DESCR'])
```

```
Optical Recognition of Handwritten Digits Data Set
===================================================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 5620
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

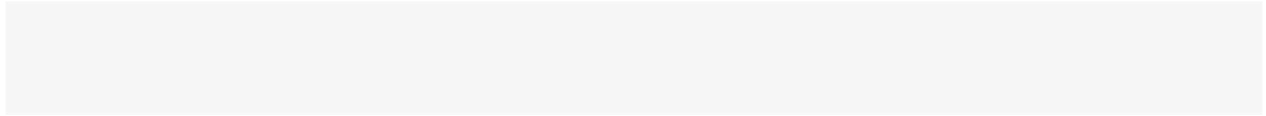
References
----------
  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

這個描述檔說明了這個資料集是在 1998年時建立的，由 `E. Alpaydin, C. Kaynak ，Department of Computer Engineering Bogazici University, Istanbul Turkey` 建立的。數字的筆跡總共來自43個人，一開始取像時爲32x32的點陣影像，之後經運算處理形成 8x8影像，其中灰階記錄的範圍則爲 0~16的整數。

(三)應用範例介紹

在整個scikit-learn應用範例中，有以下幾個範例是利用了這組手寫辨識資料集。這個資料集的使用最適合機器學習初學者來理解分類法的原理以及其進階應用

- 分類法 Classification
  - Ex 1: Recognizing hand-written digits
- 特徵選擇 Feature Selection
  - Ex 2: Recursive Feature Elimination
  - Ex 3: Recursive Feature Elimination with Cross-Validation



Datasets

機器學習資料集/ 範例三: The iris dataset

http://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

這個範例目的是介紹機器學習範例資料集中的iris 鳶尾花資料集

(一)引入函式庫及內建手寫數字資料庫

```
#這行是在ipython notebook的介面裏專用，如果在其他介面則可以拿掉
%matplotlib inline

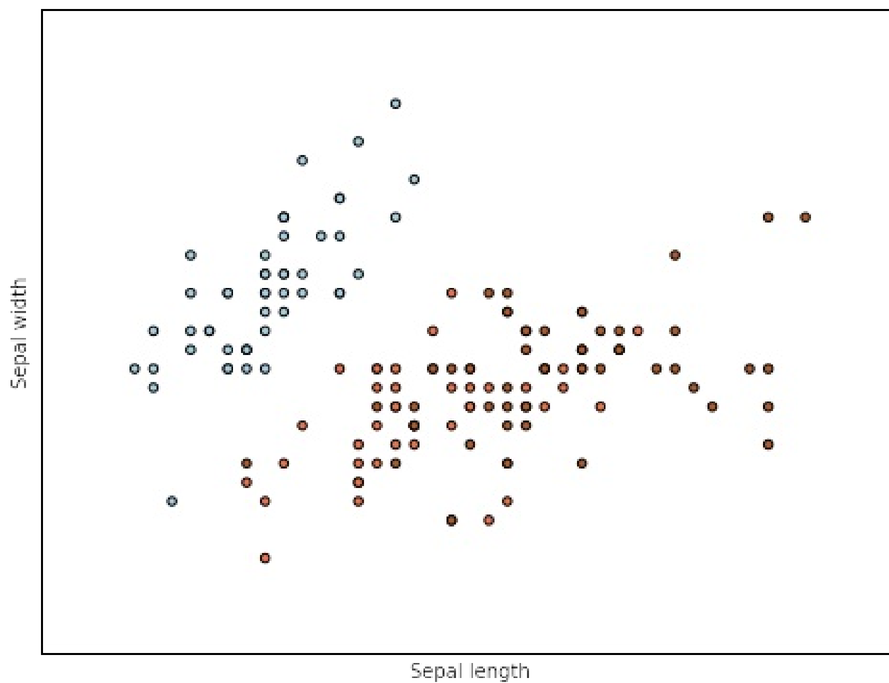
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
Y = iris.target

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

plt.figure(2, figsize=(8, 6))
plt.clf()
# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
```



(二)資料集介紹

`iris = datasets.load_iris()` 將一個dict型別資料存入iris，我們可以用下面程式碼來觀察裏面資料

```
for key,value in iris.items() :
    try:
        print (key,value.shape)
    except:
        print (key)
print(iris['feature_names'])
```

顯示	說明
('target_names', (3L,))	共有三種鳶尾花 setosa, versicolor, virginica
('data', (150L, 4L))	有150筆資料，共四種特徵
('target', (150L,))	這150筆資料各是那一種鳶尾花
DESCR	資料之描述
feature_names	四個特徵代表的意義，分別為 萼片(sepal)之長與寬以及花瓣(petal)之長與寬

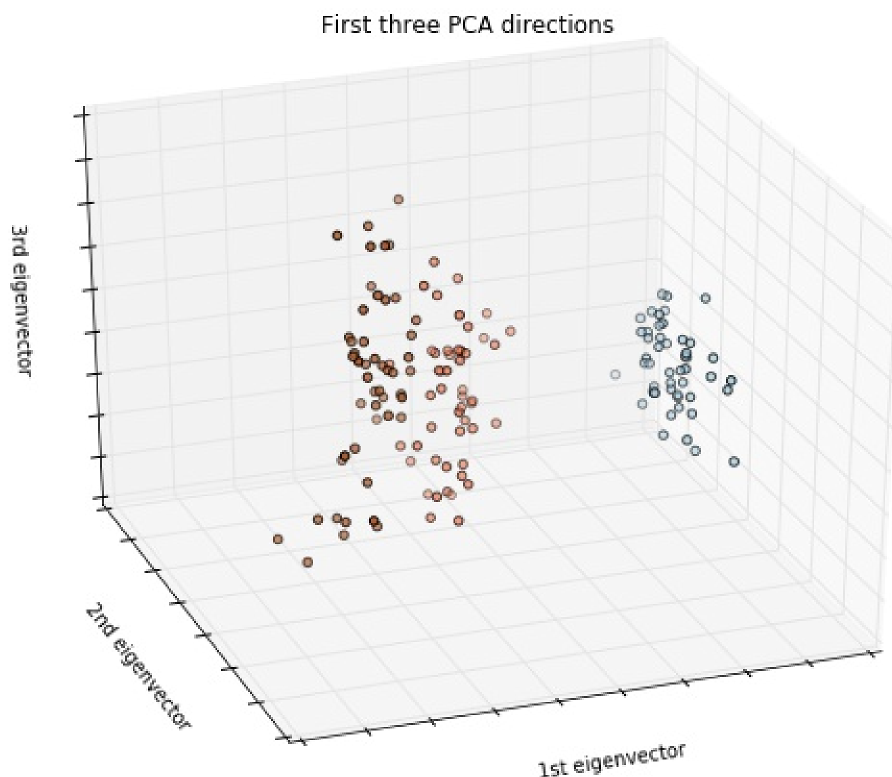
為了用視覺化方式呈現這個資料集，下面程式碼首先使用PCA演算法將資料維度降低至3

```
X_reduced = PCA(n_components=3).fit_transform(iris.data)
```

接下來將三個維度的資料立用 `mpl_toolkits.mplot3d.Axes3D` 建立三維繪圖空間，並利用 `scatter` 以三個特徵資料數值當成座標繪入空間，並以三種iris之數值 Y，來指定資料點的顏色。我們可以看出三種iris中，有一種明顯的可以與其他兩種區別，而另外兩種則無法明顯區別。

```
# To getter a better understanding of interaction of the dimensions
# plot the first three PCA dimensions
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
ax.scatter(X_reduced[:, 0], X_reduced[:, 1], X_reduced[:, 2], c=Y,
           cmap=plt.cm.Paired)
ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("2nd eigenvector")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("3rd eigenvector")
ax.w_zaxis.set_ticklabels([])

plt.show()
```




```
#接著我們嘗試將這個機器學習資料之描述檔顯示出來
print(iris['DESCR'])
```

```
Iris Plants Database

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
```

```
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20  0.76     0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML iris datasets.
http://archive.ics.uci.edu/ml/datasets/Iris

The famous Iris database, first used by Sir R.A Fisher

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

References
----------
   - Fisher,R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda,R.O., & Hart,P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

這個描述檔說明了這個資料集是在 1936年時由Fisher建立，爲圖形識別領域之重要經典範例。共例用四種特徵來分類三種鳶尾花

(三)應用範例介紹

在整個scikit-learn應用範例中，有以下幾個範例是利用了這組iris資料集。

- 分類法 Classification
  - EX 3: Plot classification probability
- 特徵選擇 Feature Selection
  - Ex 5: Test with permutations the significance of a classification score
  - Ex 6: Univariate Feature Selection
- 通用範例 General Examples
  - Ex 2: Concatenating multiple feature extraction methods

應用範例 Applications

線性回歸分析: Property value prediction

此檔案使用scikit-learn 機器學習套件裡的linear regression演算法，來達成波士頓房地產價錢預測

1. 資料集：波士頓房產
2. 特徵：房地產客觀數據，如年份、平面大小
3. 預測目標：房地產價格
4. 機器學習方法：線性迴歸
5. 探討重點：10 等分的交叉驗証(10-fold Cross-Validation)來實際測試資料以及預測值的關係
6. 關鍵函式： `sklearn.cross_validation.cross_val_predict` ； `joblib.dump` ； `joblib.load`

(一)引入函式庫及內建波士頓房地產資料庫

引入之函式庫如下

1. `sklearn.datasets` : 用來匯入內建之波士頓房地產資料庫
2. `sklearn.cross_val_predict` : 使用交叉驗證用來評估辨識準確度
3. `sklearn.linear_model` : 線性分析之模組
4. `matplotlib.pyplot` : 用來繪製影像

```
from sklearn import datasets
from sklearn.cross_validation import cross_val_predict
from sklearn import linear_model
import matplotlib.pyplot as plt

lr = linear_model.LinearRegression()
# The boston dataset
boston = datasets.load_boston()
y = boston.target
```

使用 `linear_model.LinearRegression()` 將線性迴歸分析演算法引入到 `lr` 。使用 `datasets.target` 將士頓房地產資料的預測數值匯入到 `y` 。使用 `datasets.load_boston()` 將資料存入， `boston` 爲一個dict型別資料，我們看一下資料的內容。

顯示	說明
('data', (506, 13))	房地產的資料集，共506筆房產13個特徵
('feature_names', (13,))	房地產的特徵名
('target', (506,))	回歸目標
DESCR	資料之描述

(二) `cross_val_predict` 的使用

`sklearn.cross_validation.cross_val_predict`(estimator, X, y=None, cv=None, n_jobs=1, verbose=0, fit_params=None, pre_dispatch='2*n_jobs')

X爲機器學習數據， y爲回歸目標， cv爲交叉驗証時資料切分的依據，範例爲10則將資料切分爲10等分，以其中9等分爲訓練集，另外一等分則爲測試集。

```
predicted = cross_val_predict(lr, boston.data, y, cv=10)
```

(三)使用 `joblib.dump` 匯出預測器

```
from sklearn.externals import joblib

joblib.dump(lr,"./lr_machine.pkl")
```

使用 `joblib.dump` 將線性回歸預測器匯出爲pkl檔。

(四)訓練以及分類

接著使用 `lr=joblib.load("./lr_machine.pkl")` 將pkl檔匯入爲一個linear regression預測器 `lr` 。接著使用波士頓房地產數據(boston.data)，以及預測目標(y)來訓練預測機lr `lr.fit(boston.data, y)` 。最後，使用 `predict_y=lr.predict(boston.data[2])` 預測第三筆資料的價格，並將結果存入 `predicted_y` 變數。

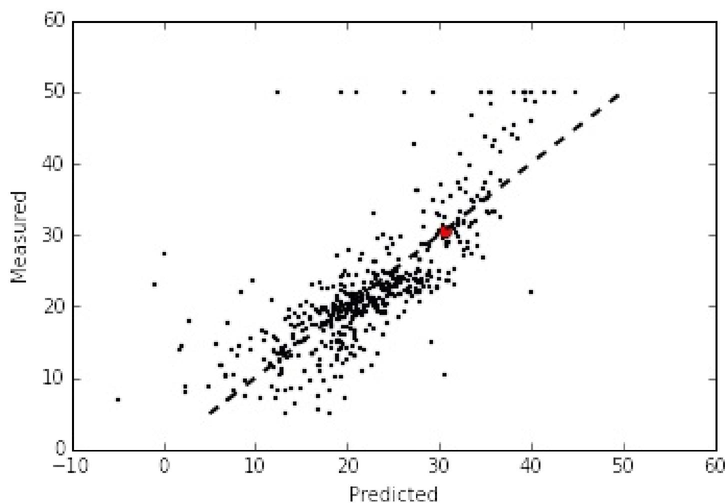
```
lr=joblib.load("./lr_machine.pkl")
lr.fit(boston.data, y)
predict_y=lr.predict(boston.data[2])
```

(五)繪出預測結果與實際目標差異圖

X軸爲預測結果，Y軸爲回歸目標。 並劃出一條斜率=1的理想曲線(用虛線標示)。

紅點爲房地產第三項數據的預測結果。

```
plt.scatter(predicted,y,s=2)
plt.plot(predict_y, predict_y, 'ro')
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)
plt.xlabel('Predicted')
plt.ylabel('Measured')
```



(六)完整程式碼

```
%matplotlib inline
from sklearn import datasets
from sklearn.cross_validation import cross_val_predict
from sklearn import linear_model
import matplotlib.pyplot as plt

lr = linear_model.LinearRegression()
boston = datasets.load_boston()
y = boston.target
# cross_val_predict returns an array of the same size as `y` where each entry
# is a prediction obtained by cross validated:
predicted = cross_val_predict(lr, boston.data, y, cv=10)
from sklearn.externals import joblib

joblib.dump(lr,"./lr_machine.pkl")
lr=joblib.load("./lr_machine.pkl")
lr.fit(boston.data, y)
predict_y=lr.predict(boston.data[2])
plt.scatter(predicted,y,s=2)
plt.plot(predict_y, predict_y, 'ro')
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)
plt.xlabel('Predicted')
plt.ylabel('Measured')
```

支持向量機回歸分析: Property value prediction

此檔案使用scikit-learn 機器學習套件裡的SVR演算法，來達成波士頓房地產價錢預測

(一)引入函式庫及內建波士頓房地產資料庫

引入之函式庫如下

1. `sklearn.datasets`：用來匯入內建之波士頓房地產資料庫
2. `sklearn.SVR`：支持向量機回歸分析之演算法
3. `matplotlib.pyplot`：用來繪製影像

```
from sklearn import datasets
from sklearn.svm import SVR
import matplotlib.pyplot as plt

boston = datasets.load_boston()
X=boston.data
y = boston.target
```

使用 `datasets.load_boston()` 將資料存入至 `boston` 。使用 `datasets.data` 將士頓房地產資料的數據資料(data)匯入到 `X` 。使用 `datasets.target` 將士頓房地產資料的預測數值匯入到 `y` 。 爲一個dict型別資料，我們看一下資料的內容。

(二) `SVR` 的使用

`sklearn.svm.SVR` (kernel='rbf', degree=3, gamma='auto', coef0=0.0, tol=0.001, C=1.0, epsilon=0.1, shrinking=True, cache_size=200, verbose=False, max_iter=-1)

```
clf = SVR(kernel='rbf', C=1e3, gamma=0.1)
clf.fit(X, y)
```

使用 `clf = SVR(kernel='rbf', C=1e3, gamma=0.1)` ，將SVR演算法引入到clf，並設定SVR演算法的參數。 使用 `clf.fit(X, y)` ，用波士頓房地產數據(boston.data)以及預測目標(y)來訓練預測機clf

(三)使用 `joblib.dump` 匯出預測器

```
from sklearn.externals import joblib
joblib.dump(clf,"./machine_SVR.pkl")
```

使用 `joblib.dump` 將SVR預測器匯出爲pkl檔。

(四)訓練以及分類

接著使用 `clf=joblib.load("./machine_SVR.pkl")` 將pkl檔匯入爲一個SVR預測器 `clf` 。接著使用波士頓房地產數據(boston.data)，以及預測目標(y)來訓練預測機clf `clf.fit(boston.data, y)` 。最後，使用 `predict_y=clf.predict(boston.data[2])` 預測第三筆資料的價格，並將結果存入 `predicted_y` 變數。

```
clf=joblib.load("./machine_SVR.pkl")
clf.fit(boston.data, y)
predict_y=clf.predict(boston.data[2])
```

(五)使用 `score` 計算準確率

先用 `predict=clf.predict(X)` 將所有波士頓房地產數據丟入clf預測機預測，並將所預測出的結果存入 `predict` 。接著使用 `clf.score(X, y)` 來計算準確率，score=1爲最理想情況，本範例中 `score` =0.99988275378631286

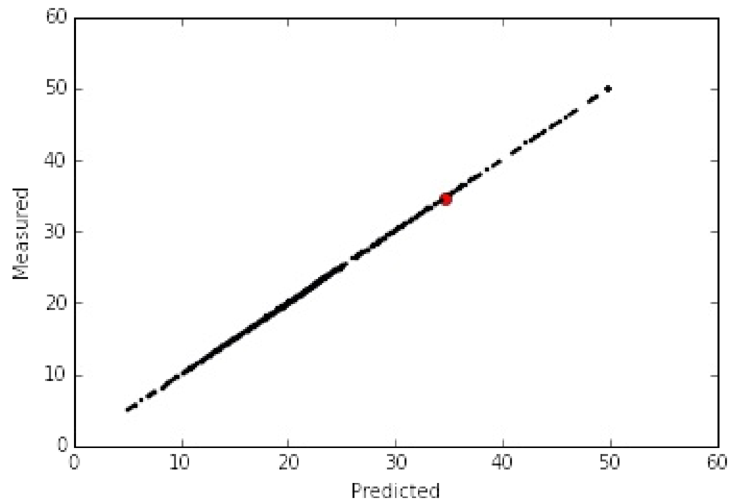
```
predict=clf.predict(X)
clf.score(X, y)
```

(六)繪出預測結果與實際目標差異圖

X軸爲預測結果，Y軸爲回歸目標。 並劃出一條斜率=1的理想曲線(用虛線標示)。 紅點爲房地產第三項數據的預測結果

因爲使用clf的準確率很高，所以預測結果與回歸目標幾乎一樣，scatter的點會幾乎都在理想曲線上。

```
plt.scatter(predict,y,s=2)
plt.plot(predict_y, predict_y, 'ro')
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)
plt.xlabel('Predicted')
plt.ylabel('Measured')
```



(六)完整程式碼

```
%matplotlib inline
from sklearn import datasets
from sklearn.svm import SVR
import matplotlib.pyplot as plt

boston = datasets.load_boston()
X=boston.data
y = boston.target
clf = SVR(kernel='rbf', C=1e3, gamma=0.1)
clf.fit(X, y)
from sklearn.externals import joblib
joblib.dump(clf,"./machine_SVR.pkl")
clf=joblib.load("./machine_SVR.pkl")
clf.fit(boston.data, y)
predict_y=clf.predict(boston.data[2])
predict=clf.predict(X)
clf.score(X, y)
plt.scatter(predict,y,s=2)
plt.plot(predict_y, predict_y, 'ro')
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)
plt.xlabel('Predicted')
plt.ylabel('Measured')
```

+

Multi-layer Perceptron(多層感知器)

http://scikit-learn.org/stable/modules/neural_networks_supervised.html

Multi-layer Perceptron (MLP):MLP爲一種監督式學習的演算法，藉由 f(·):R^m→R^o ，m是輸入時的維度、o是輸出時的維度，藉由輸入特徵 $$X=x_1,x_2,.....,x_m$$ 和目標值Y，此算法將可以使用非線性近似將資料分類或進行迴歸運算。MLP可以在輸入層與輸出層中間插入許多非線性層，如圖1所示:，這是有一層隱藏層的網路。

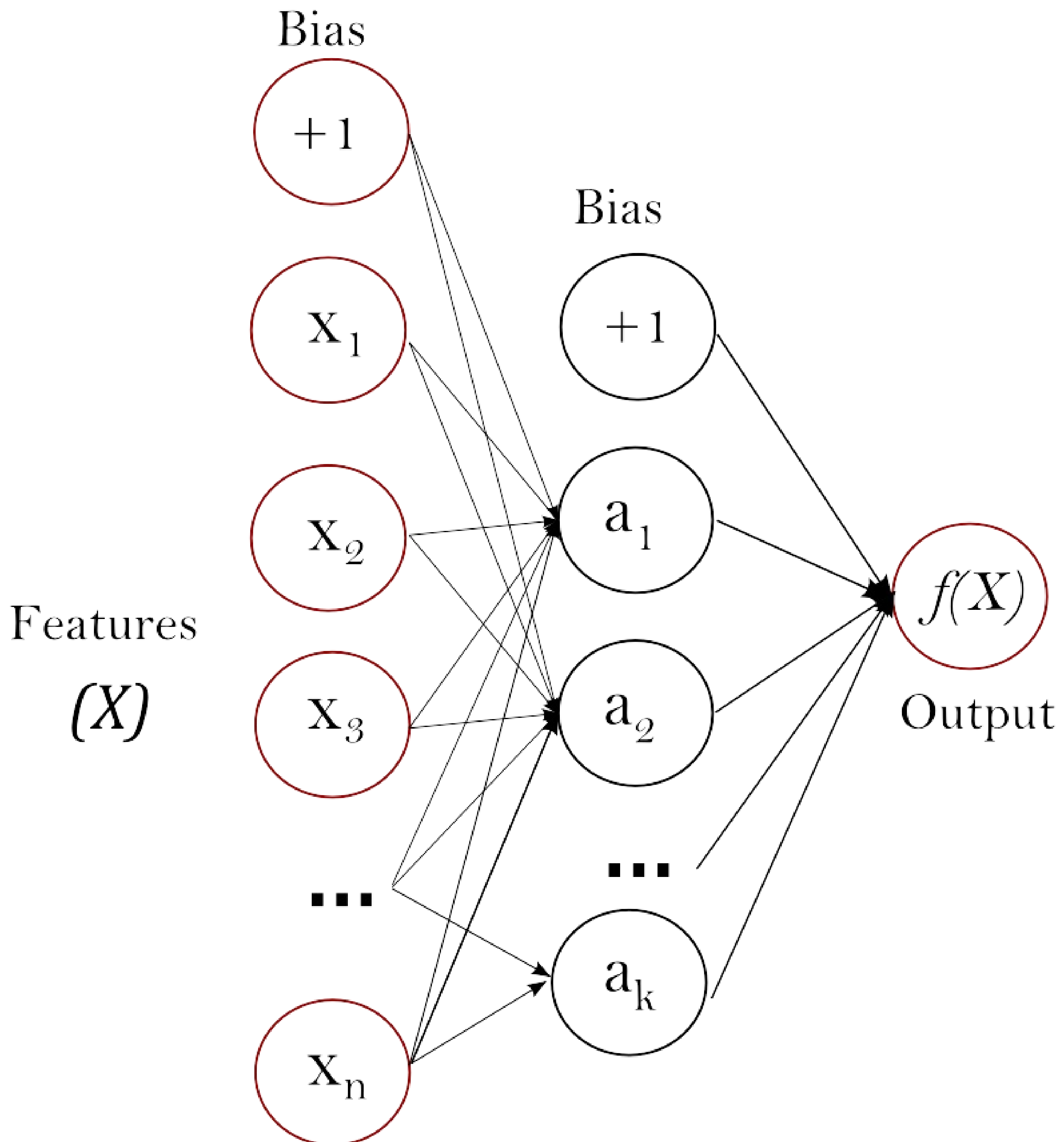


圖1:包含一層隱藏層的MLP

最左邊那層稱作輸入層，爲一個神經元集合 {x_i|x_1,x_2,...,x_m}代表輸入的特徵。每個神經元在隱藏層會根據前一層的輸出的結果，做爲此層的輸入$$w_1x_1+w_2x_2+...+w_mx_m$$在將總和使用非線性的活化函數做 f(·):R→R轉換，例如:hyperbolic tan function、Sigmoid function，最右邊那層爲輸出層，會接收最後的隱藏層的輸出在轉換一次成輸出值。

intercepts爲模型訓練後，權重矩陣內包含兩個屬性:*$coefs*\$和$intercepts_$。coefs_ 此矩陣中第i個指標表示第i層與$i+1$層的權重，intercepts_爲偏權值(bias)矩陣，此矩陣中第i個指標表示要加在$i+1層$的偏權值。

MLP優點: 1.有能力建立非線性的模型 2.可以使用$partial_fit$建立real-time模型 MLP缺點: 1.因爲凹函數擁有大於一個區域最小值，使用不同的初始權重，會讓驗證時的準確率浮動 2.MLP模型需要調整每層神經元數、層數、疊代次數 3.MLP對於特徵的預先處理很敏感，建議將特徵X都尺度降至[0,1]或[-1,+1]或讓特徵值降至平均值等於0與變異數等於1的數字區間

MLP分類器

使用MLP訓練需要使用輸入兩種陣列，一個是特徵X陣列，X陣列包含(樣本數，特徵數)，另一個是Y向量包含目標值(分類標籤)下面將會介紹MLP分類器範例。

(一)引入函式庫

from sklearn.neural_network import MLPClassifier:引進MLP分類器

(二)建立模擬資料與設定分類器參數

建立擁有三種特徵的三筆資料 X = [[0., 0.,0.], [1., 1.,1.],[2., 2.,2.]] 將三筆資料的分類標上 y = [0, 1, 2] 設定分類器:最佳化參數的演算法，alpha值，隱藏層的層數與每層神經元數: hidden_layer_sizes=(5,3)表示隱藏層有兩層第一層爲五個神經元，第二層爲三個神經元 clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5,3), random_state=1)

(三)訓練網路參數與預測

將資料丟進分類器，訓練網路參數 clf.fit(X, y)
將要預測的資料丟進網路預測 clf.predict([[2., 2., 2.], [-1., -2.,0.],[1., 1.,0.]]) 預測結果:array([2, 0, 1]) 結果表示[2., 2., 2.]爲第三類，[-1., -2.,0.]爲第一類，[1., 1.,0.]爲第二類

完整程式碼:

```
from sklearn.neural_network import MLPClassifier

X = [[0., 0.,0.], [1., 1.,1.],[2., 2.,2.]]

y = [0, 1, 2]

clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(5,3), random_state=1)

clf.fit(X, y)

clf.predict([[2., 2., 2.], [-1., -2.,0.],[1., 1.,0.]])
```

Visualization of MLP weights on MNIST

http://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html#sphx-glr-auto-examples-neural-networks-plot-mnist-filters-py

此範例將使用MNIST dataset的訓練資料集去訓練MLPClassifier，資料集中每張圖片都是28*28，對於第一層的每個神經元都會有28*28個特徵，輸出結果是將訓練資料的每個像素點對於神經元的權重畫成28*28的圖，用來表示圖片上每個像素點對於神經元的權重多寡。

(一)引入函式庫與資料

1.matplotlib.pyplot:用來繪製影像 2.sklearn.datasets:引入內建的手寫數字資料庫 3.sklearn.neural_network:引入類神經網路的套件

```
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_mldata
from sklearn.neural_network import MLPClassifier
mnist = fetch_mldata("MNIST original")
```

(二)將資料切割成訓練集與測試集

```
# 將灰階影像降尺度降到[0,1]
X, y = mnist.data / 255., mnist.target
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]
```

(三)設定分類器參數與訓練網路並畫出權重矩陣

```
#hidden_layer_sizes=(50)此處使用1層隱藏層，只有50個神經元，max_iter=10疊代訓練10次
mlp = MLPClassifier(hidden_layer_sizes=(50), max_iter=10, alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-4, random_state=1,
                    learning_rate_init=.1)

mlp.fit(X_train, y_train)
#畫出16個神經元的權重圖，黑色表示負的權重，越深色表示數值越大，白色表示正的權重，越淺色表示數值越大
fig, axes = plt.subplots(4, 4)
# use global min / max to ensure all weights are shown on the same scale
vmin, vmax = mlp.coefs_[0].min(), mlp.coefs_[0].max()
for coef, ax in zip(mlp.coefs_[0].T, axes.ravel()):
    ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray, vmin=.5 * vmin,
               vmax=.5 * vmax)
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```

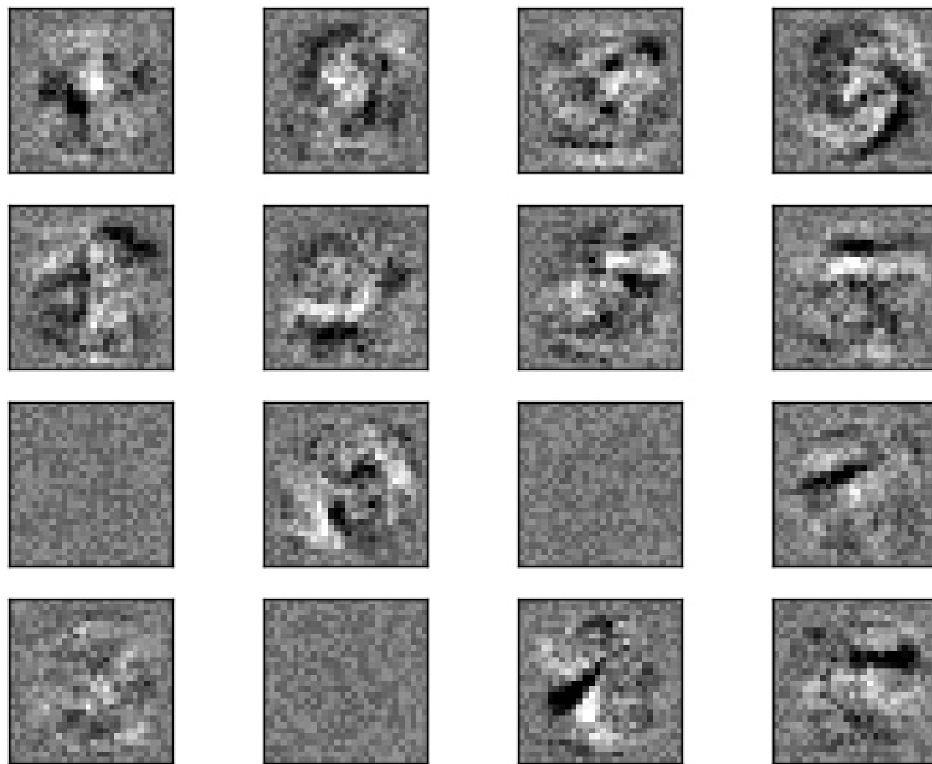


圖1:16個神經元對於影像的權重圖

```
Iteration 1, loss = 0.32212731
Iteration 2, loss = 0.15738787
Iteration 3, loss = 0.11647274
Iteration 4, loss = 0.09631113
Iteration 5, loss = 0.08074513
Iteration 6, loss = 0.07163224
Iteration 7, loss = 0.06351392
Iteration 8, loss = 0.05694146
Iteration 9, loss = 0.05213487
Iteration 10, loss = 0.04708320
```

圖2:疊代計算時loss下降

(四)完整程式碼

```
print(__doc__)

import matplotlib.pyplot as plt
from sklearn.datasets import fetch_mldata
from sklearn.neural_network import MLPClassifier

mnist = fetch_mldata("MNIST original")

X, y = mnist.data / 255., mnist.target
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]


mlp = MLPClassifier(hidden_layer_sizes=(50), max_iter=10, alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-4, random_state=1,
                    learning_rate_init=.1)

mlp.fit(X_train, y_train)
print("Training set score: %f" % mlp.score(X_train, y_train))
print("Test set score: %f" % mlp.score(X_test, y_test))

fig, axes = plt.subplots(4, 4)

vmin, vmax = mlp.coefs_[0].min(), mlp.coefs_[0].max()
for coef, ax in zip(mlp.coefs_[0].T, axes.ravel()):
    ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray, vmin=.5 * vmin,
               vmax=.5 * vmax)
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```

Restricted Boltzmann Machine features for digit classification

http://scikit-learn.org/stable/auto_examples/neural_networks/plot_rbm_logistic_classification.html#sphx-glr-auto-examples-neural-networks-plot-rbm-logistic-classification-py

此範例將使用BernoulliRBM特徵選取方法，提升手寫數字識別的精確率，伯努利限制玻爾茲曼機器模型（`BernoulliRBM

`）將可以對數據做有效的非線性 特徵提取的處理。 爲了讓此模型訓練出來更爲強健，將輸入的圖檔，分別做上左右下，一像素的平移，用以增加更多訓練資料， 訓練網路的參數是使用grid search演算法，但此訓練太耗費時間，因此不再這重現，。 此範例結果將比較， 1.使用原本的像素值做的邏輯回歸 2.使用BernoulliRBM做特徵選取的邏輯回歸 結果將顯示:使用BernoulliRBM將可以提升分類的準確度。

(一)引入函式庫與資料

```
from __future__ import print_function

print(__doc__)

# Authors: Yann N. Dauphin, Vlad Niculae, Gabriel Synnaeve
# License: BSD

import numpy as np
import matplotlib.pyplot as plt

from scipy.ndimage import convolve
from sklearn import linear_model, datasets, metrics
from sklearn.model_selection import train_test_split
from sklearn.neural_network import BernoulliRBM
from sklearn.pipeline import Pipeline
```

(二)資料前處理、讀取資料、選取模型

```
def nudge_dataset(X, Y):
    """
    此副函式是用來將輸入資料的數字圖形，分別做上左右下一像素的平移，目的是製造更多的訓練資料讓模型訓練出來更強健
    """
    direction_vectors = [
        [[0, 1, 0],
         [0, 0, 0],
         [0, 0, 0]],

        [[0, 0, 0],
         [1, 0, 0],
         [0, 0, 0]],

        [[0, 0, 0],
         [0, 0, 1],
         [0, 0, 0]],

        [[0, 0, 0],
         [0, 0, 0],
         [0, 1, 0]]]

    shift = lambda x, w: convolve(x.reshape((8, 8)), mode='constant',
                                  weights=w).ravel()
    X = np.concatenate([X] +
                       [np.apply_along_axis(shift, 1, X, vector)
                        for vector in direction_vectors])
    Y = np.concatenate([Y for _ in range(5)], axis=0)
    return X, Y

# Load Data
digits = datasets.load_digits()
X = np.asarray(digits.data, 'float32')
X, Y = nudge_dataset(X, digits.target)
X = (X - np.min(X, 0)) / (np.max(X, 0) + 0.0001)  # 將灰階影像降尺度降到[0,1]
# 將資料切割成訓練集與測試集
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                    test_size=0.2,
                                                    random_state=0)

# Models we will use
logistic = linear_model.LogisticRegression()
rbm = BernoulliRBM(random_state=0, verbose=True)

classifier = Pipeline(steps=[('rbm', rbm), ('logistic', logistic)])
```

(三)設定模型參數與訓練模型

```
# 參數選擇需使用cross-validation去比較
# 此參數是使用GridSearchCV找出來的. Here we are not performing cross-validation to save time.
#GridSratch 就是將參數設定好，跑過全部參數後去找結果最好的一組參數
rbm.learning_rate = 0.06
rbm.n_iter = 20
#.n_components = 100 表示隱藏層單元爲100，即表示萃取出100個特徵，特徵萃取的越多準確率會越高，但越耗時間
rbm.n_components = 100
logistic.C = 6000.0

# Training RBM-Logistic Pipeline
classifier.fit(X_train, Y_train)

# Training Logistic regression
logistic_classifier = linear_model.LogisticRegression(C=100.0)
logistic_classifier.fit(X_train, Y_train)
```

(四)評估模型的分辨準確率

```
print()
print("Logistic regression using RBM features:\n%s\n" % (
    metrics.classification_report(
        Y_test,
        classifier.predict(X_test))))

print("Logistic regression using raw pixel features:\n%s\n" % (
    metrics.classification_report(
        Y_test,
        logistic_classifier.predict(X_test))))
```

```
Logistic regression using RBM features:
             precision    recall  f1-score   support

          0       0.99      0.99      0.99       174
          1       0.92      0.95      0.93       184
          2       0.95      0.98      0.97       166
          3       0.97      0.91      0.94       194
          4       0.97      0.95      0.96       186
          5       0.93      0.93      0.93       181
          6       0.98      0.97      0.97       207
          7       0.95      1.00      0.97       154
          8       0.90      0.88      0.89       182
          9       0.91      0.93      0.92       169

avg / total       0.95      0.95      0.95      1797


Logistic regression using raw pixel features:
             precision    recall  f1-score   support
```

圖1:使用RBM演算法後準確率爲0.95

```
Logistic regression using raw pixel features:
             precision    recall  f1-score   support

          0       0.85      0.94      0.89       174
          1       0.57      0.55      0.56       184
          2       0.72      0.85      0.78       166
          3       0.76      0.74      0.75       194
          4       0.85      0.82      0.84       186
          5       0.74      0.75      0.75       181
          6       0.93      0.88      0.91       207
          7       0.86      0.90      0.88       154
          8       0.68      0.55      0.61       182
          9       0.71      0.74      0.72       169

avg / total       0.77      0.77      0.77      1797
```

圖2:不使用任何特徵選取方法做的做的邏輯回歸準確率0.77

(五)畫出100個RBM萃取出的特徵

```
plt.figure(figsize=(4.2, 4))
for i, comp in enumerate(rbm.components_):
    plt.subplot(10, 10, i + 1)
    plt.imshow(comp.reshape((8, 8)), cmap=plt.cm.gray_r,
               interpolation='nearest')
    plt.xticks(())
    plt.yticks(())
plt.suptitle('100 components extracted by RBM', fontsize=16)
plt.subplots_adjust(0.08, 0.02, 0.92, 0.85, 0.08, 0.23)

plt.show()
```



圖3:使用RBM演算法，尋找出來的特徵

(六)完整程式碼

```
from __future__ import print_function

print(__doc__)

# Authors: Yann N. Dauphin, Vlad Niculae, Gabriel Synnaeve
# License: BSD
```

```
import numpy as np
import matplotlib.pyplot as plt

from scipy.ndimage import convolve
from sklearn import linear_model, datasets, metrics
from sklearn.model_selection import train_test_split
from sklearn.neural_network import BernoulliRBM
from sklearn.pipeline import Pipeline


###############################################################################
# Setting up

def nudge_dataset(X, Y):
    """
    This produces a dataset 5 times bigger than the original one,
    by moving the 8x8 images in X around by 1px to left, right, down, up
    """
    direction_vectors = [
        [[0, 1, 0],
         [0, 0, 0],
         [0, 0, 0]],

        [[0, 0, 0],
         [1, 0, 0],
         [0, 0, 0]],

        [[0, 0, 0],
         [0, 0, 1],
         [0, 0, 0]],

        [[0, 0, 0],
         [0, 0, 0],
         [0, 1, 0]]]

    shift = lambda x, w: convolve(x.reshape((8, 8)), mode='constant',
                                  weights=w).ravel()
    X = np.concatenate([X] +
                       [np.apply_along_axis(shift, 1, X, vector)
                        for vector in direction_vectors])
    Y = np.concatenate([Y for _ in range(5)], axis=0)
    return X, Y

# Load Data
digits = datasets.load_digits()
X = np.asarray(digits.data, 'float32')
X, Y = nudge_dataset(X, digits.target)
X = (X - np.min(X, 0)) / (np.max(X, 0) + 0.0001)  # 0-1 scaling

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                    test_size=0.2,
                                                    random_state=0)
```

```
# Models we will use
logistic = linear_model.LogisticRegression()
rbm = BernoulliRBM(random_state=0, verbose=True)

classifier = Pipeline(steps=[('rbm', rbm), ('logistic', logistic)])

###############################################################################
# Training

# Hyper-parameters. These were set by cross-validation,
# using a GridSearchCV. Here we are not performing cross-validation to
# save time.
rbm.learning_rate = 0.06
rbm.n_iter = 20
# More components tend to give better prediction performance, but larger
# fitting time
rbm.n_components = 100
logistic.C = 6000.0

# Training RBM-Logistic Pipeline
classifier.fit(X_train, Y_train)

# Training Logistic regression
logistic_classifier = linear_model.LogisticRegression(C=100.0)
logistic_classifier.fit(X_train, Y_train)

###############################################################################
# Evaluation

print()
print("Logistic regression using RBM features:\n%s\n" % (
    metrics.classification_report(
        Y_test,
        classifier.predict(X_test))))

print("Logistic regression using raw pixel features:\n%s\n" % (
    metrics.classification_report(
        Y_test,
        logistic_classifier.predict(X_test))))

###############################################################################
# Plotting

plt.figure(figsize=(4.2, 4))
for i, comp in enumerate(rbm.components_):
    plt.subplot(10, 10, i + 1)
    plt.imshow(comp.reshape((8, 8)), cmap=plt.cm.gray_r,
               interpolation='nearest')
    plt.xticks(())
    plt.yticks(())
plt.suptitle('100 components extracted by RBM', fontsize=16)
plt.subplots_adjust(0.08, 0.02, 0.92, 0.85, 0.08, 0.23)
```

```
plt.show()
```

Compare Stochastic learning strategies for MLPClassifier

http://scikit-learn.org/stable/auto_examples/neural_networks/plot_mlp_training_curves.html#sphx-glr-auto-examples-neural-networks-plot-mlp-training-curves-py

此範例將畫出圖表，展現不同的訓練策略(optimizer)下loss curves的變化，訓練策略包括SGD與Adam。

1.Stochastic Gradient Descent(SGD):

.Stochastic Gradient Descent(SGD)爲Gradient Descent(GD)的改良，在GD裡是輸入全部的training dataset，根據累積的loss才更新一次權重，因此收斂速度很慢，SGD隨機抽一筆 training sample，依照其 loss 更新權重。

2.Momentum:

Momentum是爲了以防GD類的方法陷入局部最小值而衍生的方法，可以利用momentum降低陷入local minimum的機率，此方法是參考物理學動量的觀念。

看圖1藍色點的位置，當GD類的方法陷入局部最小值時，因爲gd=0將會使電腦認爲此處爲最小值，於是爲了減少此現象，每次更新時會將上次更新權重的一部分拿來加入此次更新。如紅色箭頭所示，將有機會翻過local minimum。

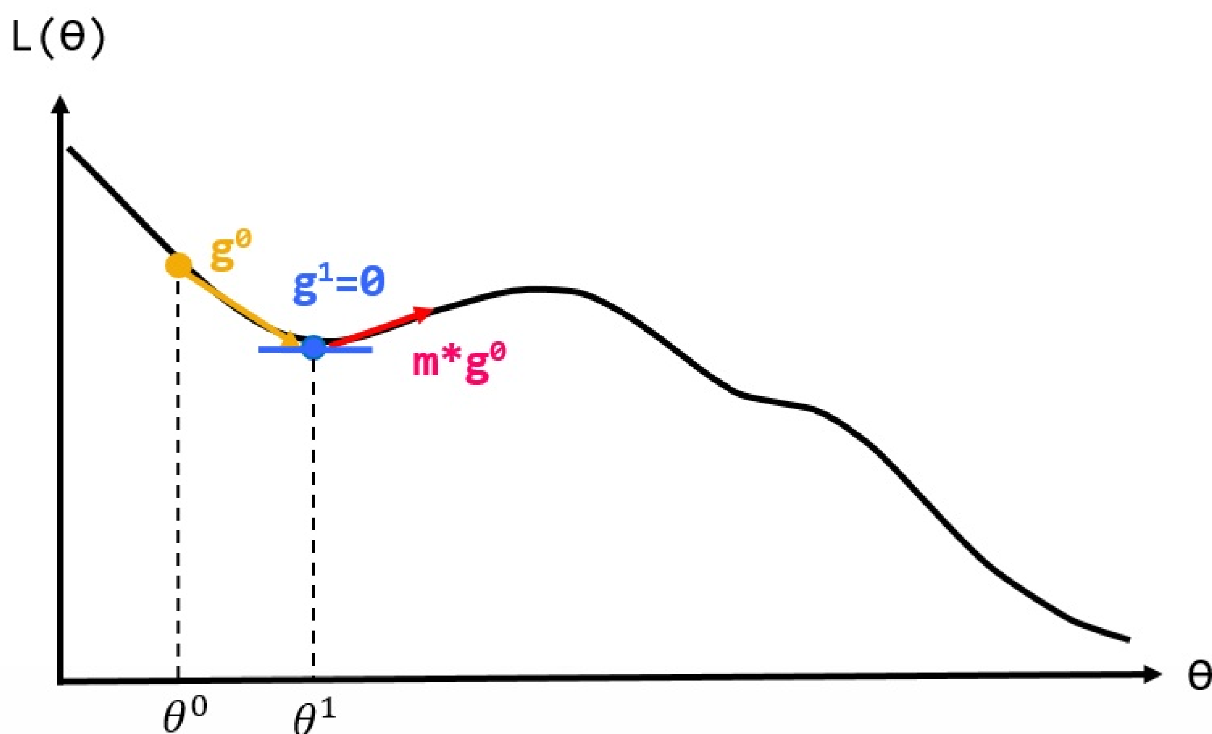


圖1:momentum觀念示意圖

3.Nesterov Momentum:

Nesterov Momentum爲另外一種Momentum的變形體，目的也是降低陷入local minimum機率的方法，而兩種方法的差異在於下圖:

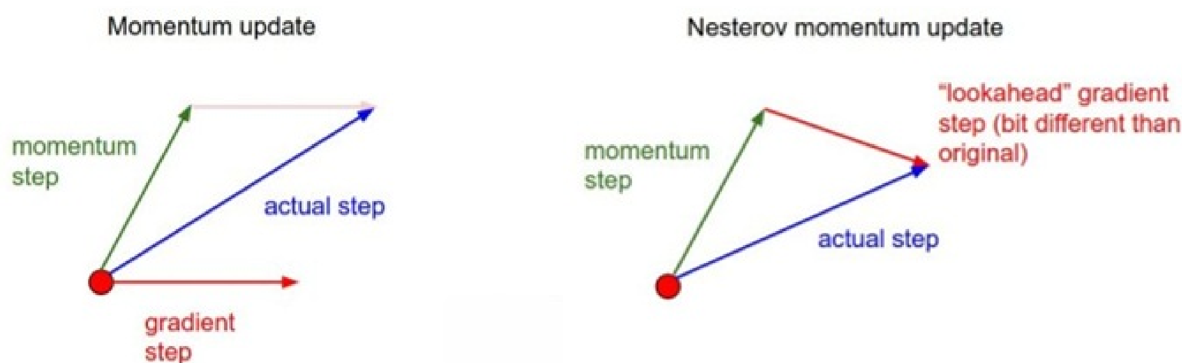


圖2:左圖爲momentum，1.先計算 gradient、2.加上 momentum、3.更新權重 右圖爲Nesterov Momentum，1.先加上 momentum、2.計算gradient、3.更新權重。

圖2圖片來源:http://cs231n.github.io/neural-networks-3/

4.Adaptive Moment Estimation (Adam):

Adam爲一種自己更新學習速率的方法，會根據GD計算出來的值調整每個參數的學習率(因材施教)。 以上所有的最佳化方法都將需要設定learning_rate_init值，此範例結果將呈現四種不同資料的比較:iris資料集、digits資料集、與使用sklearn.datasets產生資料集circles、 moon。

(一)引入函式庫

```
print(__doc__)
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn import datasets
```

(二)設定模型參數

```
# different learning rate schedules and momentum parameters
params = [{'solver': 'sgd', 'learning_rate': 'constant', 'momentum': 0,
           'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'constant', 'momentum': .9,
           'nesterovs_momentum': False, 'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'constant', 'momentum': .9,
           'nesterovs_momentum': True, 'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum': 0,
           'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum': .9,
           'nesterovs_momentum': True, 'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum': .9,
           'nesterovs_momentum': False, 'learning_rate_init': 0.2},
          {'solver': 'adam', 'learning_rate_init': 0.01}]

labels = ["constant learning-rate", "constant with momentum",
          "constant with Nesterov's momentum",
          "inv-scaling learning-rate", "inv-scaling with momentum",
          "inv-scaling with Nesterov's momentum", "adam"]

plot_args = [{'c': 'red', 'linestyle': '-'},
             {'c': 'green', 'linestyle': '-'},
             {'c': 'blue', 'linestyle': '-'},
             {'c': 'red', 'linestyle': '--'},
             {'c': 'green', 'linestyle': '--'},
             {'c': 'blue', 'linestyle': '--'},
             {'c': 'black', 'linestyle': '-'}]
```

(三)畫出loss curves

```
def plot_on_dataset(X, y, ax, name):
    # for each dataset, plot learning for each learning strategy
    print("\nlearning on dataset %s" % name)
    ax.set_title(name)
    X = MinMaxScaler().fit_transform(X)
    mlps = []
    if name == "digits":
        # digits is larger but converges fairly quickly
        max_iter = 15
    else:
        max_iter = 400

    for label, param in zip(labels, params):
        print("training: %s" % label)
        mlp = MLPClassifier(verbose=0, random_state=0,
                            max_iter=max_iter, **param)
        mlp.fit(X, y)
        mlps.append(mlp)
        print("Training set score: %f" % mlp.score(X, y))
        print("Training set loss: %f" % mlp.loss_)
    for mlp, label, args in zip(mlps, labels, plot_args):
            ax.plot(mlp.loss_curve_, label=label, **args)


fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# load / generate some toy datasets
iris = datasets.load_iris()
digits = datasets.load_digits()
data_sets = [(iris.data, iris.target),
             (digits.data, digits.target),
             datasets.make_circles(noise=0.2, factor=0.5, random_state=1),
             datasets.make_moons(noise=0.3, random_state=0)]

for ax, data, name in zip(axes.ravel(), data_sets, ['iris', 'digits',
                                                    'circles', 'moons']):
    plot_on_dataset(*data, ax=ax, name=name)

fig.legend(ax.get_lines(), labels=labels, ncol=3, loc="upper center")
plt.show()
```

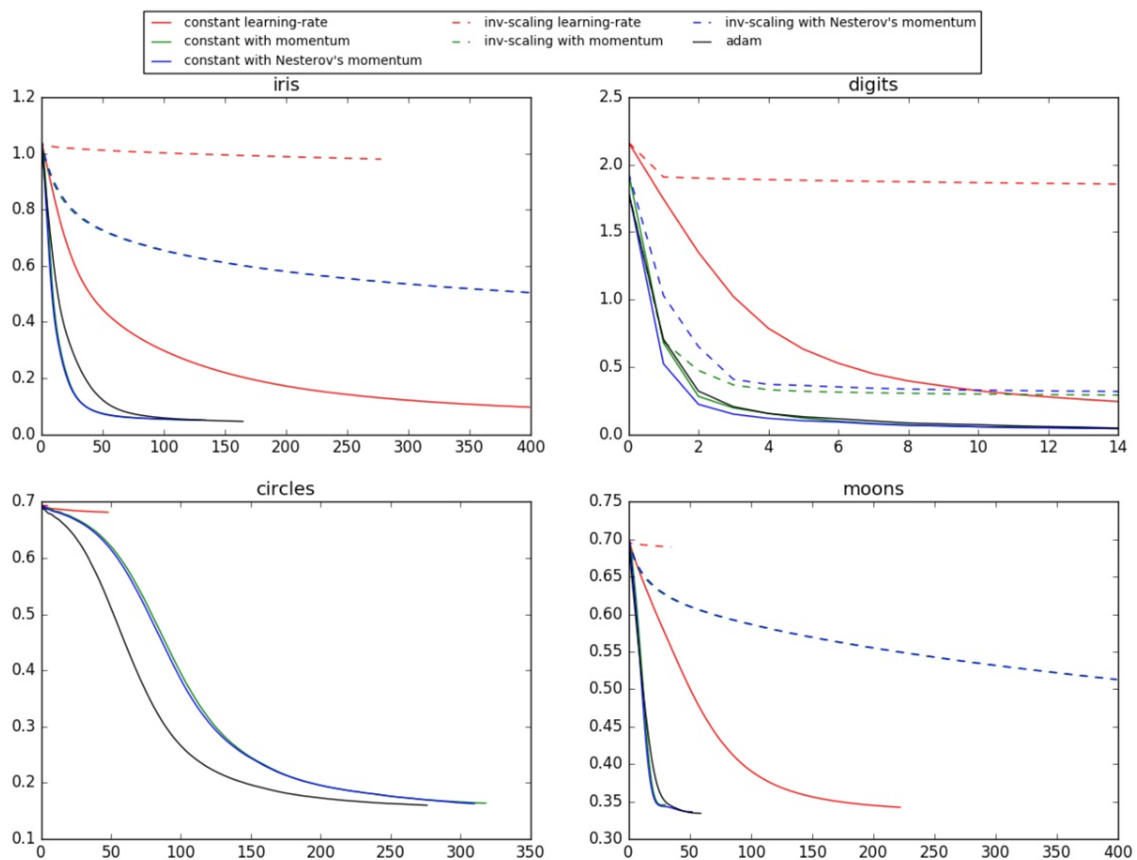


圖3:四種資料對於不同學習方法的loss curves下降比較圖

(四)完整程式碼

```
print(__doc__)
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn import datasets

# different learning rate schedules and momentum parameters
params = [{'solver': 'sgd', 'learning_rate': 'constant', 'momentum': 0,
           'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'constant', 'momentum': .9,
           'nesterovs_momentum': False, 'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'constant', 'momentum': .9,
           'nesterovs_momentum': True, 'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum': 0,
           'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum': .9,
           'nesterovs_momentum': True, 'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum': .9,
           'nesterovs_momentum': False, 'learning_rate_init': 0.2},
          {'solver': 'adam', 'learning_rate_init': 0.01}]

labels = ["constant learning-rate", "constant with momentum",
          "constant with Nesterov's momentum",
          "inv-scaling learning-rate", "inv-scaling with momentum",
```

```
            "inv-scaling with Nesterov's momentum", "adam"]

plot_args = [{'c': 'red', 'linestyle': '-'},
             {'c': 'green', 'linestyle': '-'},
             {'c': 'blue', 'linestyle': '-'},
             {'c': 'red', 'linestyle': '--'},
             {'c': 'green', 'linestyle': '--'},
             {'c': 'blue', 'linestyle': '--'},
             {'c': 'black', 'linestyle': '-'}]


def plot_on_dataset(X, y, ax, name):
    # for each dataset, plot learning for each learning strategy
    print("\nlearning on dataset %s" % name)
    ax.set_title(name)
    X = MinMaxScaler().fit_transform(X)
    mlps = []
    if name == "digits":
        # digits is larger but converges fairly quickly
        max_iter = 15
    else:
        max_iter = 400

    for label, param in zip(labels, params):
        print("training: %s" % label)
        mlp = MLPClassifier(verbose=0, random_state=0,
                            max_iter=max_iter, **param)
        mlp.fit(X, y)
        mlps.append(mlp)
        print("Training set score: %f" % mlp.score(X, y))
        print("Training set loss: %f" % mlp.loss_)
    for mlp, label, args in zip(mlps, labels, plot_args):
            ax.plot(mlp.loss_curve_, label=label, **args)


fig, axes = plt.subplots(2, 2, figsize=(15, 10))
# load / generate some toy datasets
iris = datasets.load_iris()
digits = datasets.load_digits()
data_sets = [(iris.data, iris.target),
             (digits.data, digits.target),
             datasets.make_circles(noise=0.2, factor=0.5, random_state=1),
             datasets.make_moons(noise=0.3, random_state=0)]

for ax, data, name in zip(axes.ravel(), data_sets, ['iris', 'digits',
                                                    'circles', 'moons']):
    plot_on_dataset(*data, ax=ax, name=name)

fig.legend(ax.get_lines(), labels=labels, ncol=3, loc="upper center")
plt.show()
```

Varying regularization in Multi-layer Perceptron

http://scikit-learn.org/stable/auto_examples/neural_networks/plot_mlp_alpha.html#sphx-glr-auto-examples-neural-networks-plot-mlp-alpha-py

此範例是比較不同的正歸化參數'alpha'，對於使用scikit-learn的資料產生器 ，所產生的circles、 moon 和random n-class classification，三種資料集的成效。 PS:正規化爲一種處理無限大、發散以及一些不合理表示式的方法，透過引入一項輔助性的概念——正規子(regulator)，去限制函數使得函數不會發散 此處的Alpha參數即爲正規子，目的是去限制權重(Weight,W)的大小，以防萬一overfitting與underfitting的問題，增加alpha值可能可以處理overfitting，反之減小alpha可能可以解決underfitting的問題，至於權重大小，如何影響輸出請看圖1:

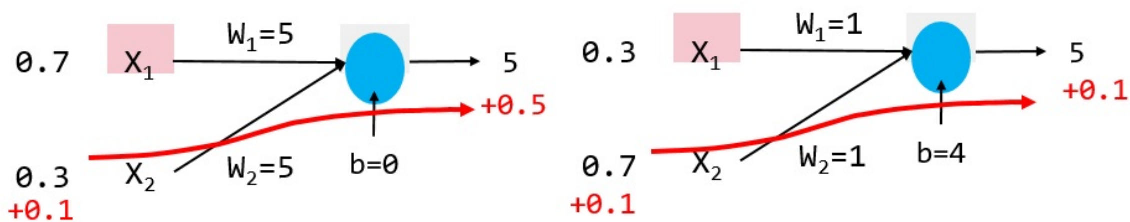


圖1:比較同樣輸入，對於不同大小權重值，對於輸出的影響左圖爲權重爲5時，當輸入變動0.1時，輸出增加0.5，即輸出改變10%，右圖爲權重爲1時，當輸入變動0.1時，輸出增加0.1，即輸出改變2%，通常模型對於input較不敏感，模型表現較好 結果將顯示出:使用不同alpha值去限制權重產生出的決策邊界

(一)引入函式庫

```
print(__doc__)


# Author: Issam H. Laradji
# License: BSD 3 clause

import numpy as np
from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
```

(二)設定模型參數與產生資料

```
h = .02  # step size in the mesh

alphas = np.logspace(-5, 3, 5)#
names = []
for i in alphas:
    names.append('alpha ' + str(i))

classifiers = []
for i in alphas:
    classifiers.append(MLPClassifier(alpha=i, random_state=1))

X, y = make_classification(n_features=2, n_redundant=0, n_informative=2,
                           random_state=0, n_clusters_per_class=1)
rng = np.random.RandomState(2)
X += 2 * rng.uniform(size=X.shape)
linearly_separable = (X, y)

datasets = [make_moons(noise=0.3, random_state=0),
            make_circles(noise=0.2, factor=0.5, random_state=1),
            linearly_separable]

figure = plt.figure(figsize=(17, 9))
i = 1
# iterate over datasets
for X, y in datasets:
    # preprocess dataset, split into training and test part
    X = StandardScaler().fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4)

    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
```

(三)繪製圖形

```
    # just plot the dataset first
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])
    ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
    # Plot the training points
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
    # and testing points
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6)
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    i += 1

    # iterate over classifiers
    for name, clf in zip(names, classifiers):
        ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
        clf.fit(X_train, y_train)
        score = clf.score(X_test, y_test)

        # Plot the decision boundary. For that, we will assign a color to each
        # point in the mesh [x_min, x_max]x[y_min, y_max].
        if hasattr(clf, "decision_function"):
            Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
        else:
            Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]

        # Put the result into a color plot
        Z = Z.reshape(xx.shape)
        ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)

        # Plot also the training points
        ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
        # and testing points
        ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright,
                   alpha=0.6)

        ax.set_xlim(xx.min(), xx.max())
        ax.set_ylim(yy.min(), yy.max())
        ax.set_xticks(())
        ax.set_yticks(())
        ax.set_title(name)
        ax.text(xx.max() - .3, yy.min() + .3, ('%.2f' % score).lstrip('0'),
                size=15, horizontalalignment='right')
        i += 1

figure.subplots_adjust(left=.02, right=.98)
plt.show()
```

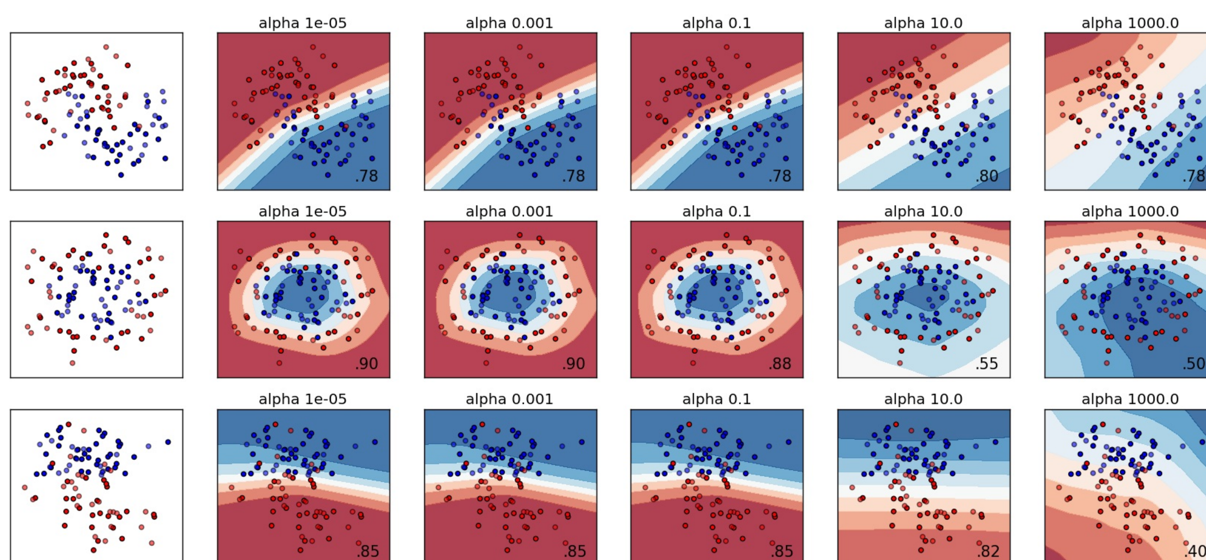


圖2:不同alpha結果圖，每張子圖右下角是分辨率，alpha值很大，模型的結果明顯underfitting

決策樹 Decision trees

決策樹/範例一: Decision Tree Regression

http://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html#sphx-glr-auto-examples-tree-plot-tree-regression-py

範例目的

此範例利用Decision Tree從數據中學習一組if-then-else決策規則，逼近加有雜訊的sine curve，因此它模擬出局部的線性迴歸以近似sine curve。 若決策樹深度越深(可由max_depth參數控制)，則決策規則越複雜，模型也會越接近數據，但若數據中含有雜訊，太深的樹就有可能產生過擬合的情形。 此範例模擬了不同深度的樹，當用帶有雜點的數據可能造成的情況。

(一)引入函式庫及建立隨機數據資料

引入函式資料庫

- `matplotlib.pyplot` ：用來繪製影像。
- `sklearn.tree import DecisionTreeRegressor` ：利用決策樹方式建立預測模型。

特徵資料

- `np.random()` ：隨機產生介於0~1之間的亂數
- `RandomState.rand(d0,d1,..,dn)` ：給定隨機亂數的矩陣形狀
- `np.sort` 將資料依大小排序。

目標資料

- `np.sin(X)` ：以X做爲徑度，計算出相對的sine值。
- `ravel()` ：輸出連續的一維矩陣。
- `y[::5] += 3 * (0.5 - rng.rand(16))` ：爲目標資料加入雜訊點。

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt

rng = np.random.RandomState(1)
X = np.sort(5* rng.rand(80, 1), axis=0)  #0~5之間隨機產生80個數值

y = np.sin(X).ravel()
y[::5] += 3 * (0.5 - rng.rand(16)) #每5筆資料加入一個雜訊
```

(二)建立Decision Tree迴歸模型

建立模型

- `DecisionTreeRegressor(max_depth = 最大深度)` ： `DecisionTreeRegressor` 建立決策樹回歸模型。 `max_depth` 決定樹的深度，若爲None則所有節點被展開。此範例會呈現不同 `max_depth` 對預測結果的影響。

模型訓練

- `fit(特徵資料, 目標資料)` ：利用特徵資料及目標資料對迴歸模型進行訓練。

預測結果

- `np.arrange(起始點, 結束點, 間隔)`：`np.arange(0.0, 5.0, 0.01)` 在0~5之間每0.01取一格，建立預測輸入點矩陣。
- `np.newaxis`：增加矩陣維度。
- `predict(輸入矩陣)`：對訓練完畢的模型測試，輸出爲預測結果。

```
regr_1 = DecisionTreeRegressor(max_depth=2) #最大深度爲2的決策樹
regr_2 = DecisionTreeRegressor(max_depth=5) #最大深度爲5的決策樹

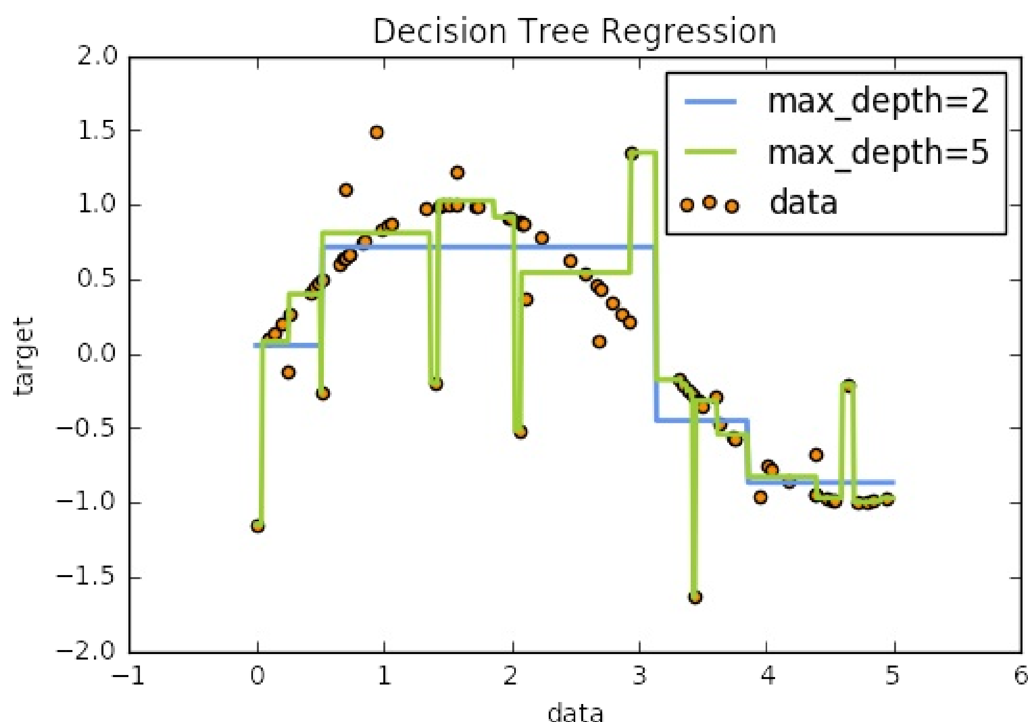
regr_1.fit(X, y)
regr_2.fit(X, y)

X_test = np.arange(0.0, 5.0, 0.01)[:, np.newaxis]
y_1 = regr_1.predict(X_test)
y_2 = regr_2.predict(X_test)
```

(三) 繪出預測結果與實際目標圖

- `plt.scatter(X,y)`：將X、y以點的方式繪製於平面上，c爲數據點的顏色，label爲圖例。
- `plt.plot(X,y)`：將X、y以連線方式繪製於平面上，color爲線的顏色，label爲圖例，linewidth爲線的寬度。

```
plt.figure()
plt.scatter(X, y, c="darkorange", label="data")
plt.plot(X_test, y_1, color="cornflowerblue", label="max_depth=2", linewidth=2)
plt.plot(X_test, y_2, color="yellowgreen", label="max_depth=5", linewidth=2)
plt.xlabel("data") #x軸代表data數值
plt.ylabel("target") #y軸代表target數值
plt.title("Decision Tree Regression") #標示圖片的標題
plt.legend() #繪出圖例
plt.show()
```



(四)完整程式碼

```
print(__doc__)

# Import the necessary modules and libraries
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt

# Create a random dataset
rng = np.random.RandomState(1)
X = np.sort(5 * rng.rand(80, 1), axis=0)
y = np.sin(X).ravel()
y[::5] += 3 * (0.5 - rng.rand(16))

# Fit regression model
regr_1 = DecisionTreeRegressor(max_depth=2)
regr_2 = DecisionTreeRegressor(max_depth=5)
regr_1.fit(X, y)
regr_2.fit(X, y)

# Predict
X_test = np.arange(0.0, 5.0, 0.01)[:, np.newaxis]
y_1 = regr_1.predict(X_test)
y_2 = regr_2.predict(X_test)

# Plot the results
plt.figure()
plt.scatter(X, y, c="darkorange", label="data")
plt.plot(X_test, y_1, color="cornflowerblue", label="max_depth=2", linewidth=2)
plt.plot(X_test, y_2, color="yellowgreen", label="max_depth=5", linewidth=2)
plt.xlabel("data")
plt.ylabel("target")
plt.title("Decision Tree Regression")
plt.legend()
plt.show()
```

決策樹/範例二:Multi-output Decision Tree Regression

http://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression_multioutput.html#sphx-glr-auto-examples-tree-plot-tree-regression-multioutput-py

範例目的

此範例用決策樹說明多輸出迴歸的例子，利用帶有雜訊的特徵及目標值模擬出近似圓的局部線性迴歸。 若決策樹深度越深(可由max_depth參數控制)，則決策規則越複雜，模型也會越接近數據，但若數據中含有雜訊，太深的樹就有可能產生過擬合的情形。 此範例模擬了不同深度的樹，當用帶有雜點的數據可能造成的情況。

(一)引入函式庫及建立隨機數據資料

引入函式資料庫

- `matplotlib.pyplot` ：用來繪製影像。
- `sklearn.tree import DecisionTreeRegressor` ：利用決策樹方式建立預測模型。

特徵資料

- `np.random()` ：隨機產生介於0~1之間的亂數
- `RandomState.rand(d0,d1,..,dn)` ：給定隨機亂數的矩陣形狀
- `np.sort` 將資料依大小排序。

目標資料

- `np.sin(X)` ：以X做為徑度，計算出相對的sine值。
- `ravel()` ：輸出連續的一維矩陣。
- `y[::5, :] += (0.5 - rng.rand(20, 2))` ：為目標資料加入雜訊點。

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt

rng = np.random.RandomState(1)
X = np.sort(200 * rng.rand(100, 1) - 100, axis=0) #在-100~100之間隨機建立100個點

y = np.array([np.pi * np.sin(X).ravel(), np.pi * np.cos(X).ravel()]).T #每個X產生兩個輸出分別為sine及cosine值，並存於y中
y[::5, :] += (0.5 - rng.rand(20, 2)) #每5筆資料加入一個雜訊
```

(二)建立Decision Tree迴歸模型

建立模型

- `DecisionTreeRegressor(max_depth = 最大深度)` ： `DecisionTreeRegressor` 建立決策樹回歸模型。 `max_depth` 決定樹的深度，若為None則所有節點被展開。此範例會呈現不同 `max_depth` 對預測結果的影響。

模型訓練

- `fit(特徵資料, 目標資料)` ：利用特徵資料及目標資料對迴歸模型進行訓練。

預測結果

- `np.arrange(起始點, 結束點, 間隔)` : `np.arange(-100.0, 100.0, 0.01)` 在-100~100之間每0.01取一格，建立預測輸入點矩陣。
- `np.newaxis` ：增加矩陣維度。
- `predict(輸入矩陣)` ：對訓練完畢的模型測試，輸出爲預測結果。

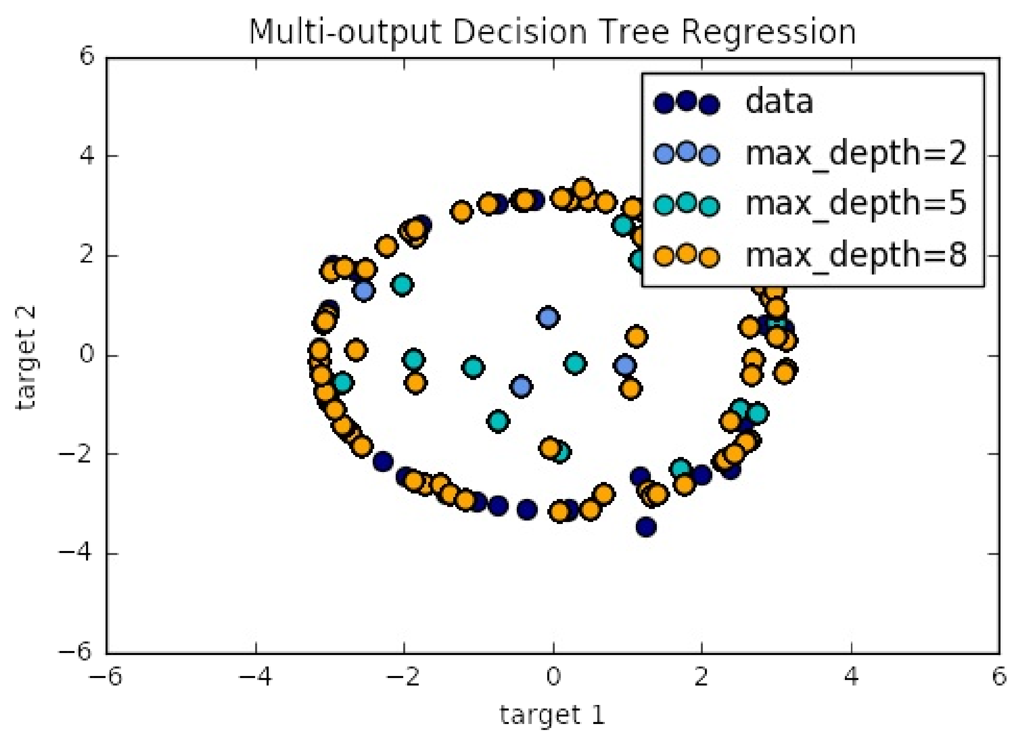
```
# Fit regression model
regr_1 = DecisionTreeRegressor(max_depth=2) #最大深度爲2的決策樹
regr_2 = DecisionTreeRegressor(max_depth=5) #最大深度爲5的決策樹
regr_3 = DecisionTreeRegressor(max_depth=8) #最大深度爲8的決策樹
regr_1.fit(X, y)
regr_2.fit(X, y)
regr_3.fit(X, y)

# Predict
X_test = np.arange(-100.0, 100.0, 0.01)[:, np.newaxis]
y_1 = regr_1.predict(X_test)
y_2 = regr_2.predict(X_test)
y_3 = regr_3.predict(X_test)
```

(三) 繪出預測結果與實際目標圖

- `plt.scatter(X,y)` ：將X、y以點的方式繪製於平面上，c爲數據點的顏色，s決定點的大小，label爲圖例。

```
plt.figure()
s = 50
plt.scatter(y[:, 0], y[:, 1], c="navy", s=s, label="data")
plt.scatter(y_1[:, 0], y_1[:, 1], c="cornflowerblue", s=s, label="max_depth=2")
plt.scatter(y_2[:, 0], y_2[:, 1], c="c", s=s, label="max_depth=5")
plt.scatter(y_3[:, 0], y_3[:, 1], c="orange", s=s, label="max_depth=8")
plt.xlim([-6, 6]) #設定x軸的上下限
plt.ylim([-6, 6]) #設定y軸的上下限
plt.xlabel("target 1") #x軸代表target 1數值
plt.ylabel("target 2") #x軸代表target 2數值
plt.title("Multi-output Decision Tree Regression") #標示圖片的標題
plt.legend() #繪出圖例
plt.show()
```


Multi-output Decision Tree Regression
data
max_depth=2
max_depth=5
max_depth=8
target 1
target 2
−6
−4
−2
0
2
4
6

(四)完整程式碼

```
print(__doc__)

import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor

# Create a random dataset
rng = np.random.RandomState(1)
X = np.sort(200 * rng.rand(100, 1) - 100, axis=0)
y = np.array([np.pi * np.sin(X).ravel(), np.pi * np.cos(X).ravel()]).T
y[::5, :] += (0.5 - rng.rand(20, 2))

# Fit regression model
regr_1 = DecisionTreeRegressor(max_depth=2)
regr_2 = DecisionTreeRegressor(max_depth=5)
regr_3 = DecisionTreeRegressor(max_depth=8)
regr_1.fit(X, y)
regr_2.fit(X, y)
regr_3.fit(X, y)

# Predict
X_test = np.arange(-100.0, 100.0, 0.01)[:, np.newaxis]
y_1 = regr_1.predict(X_test)
y_2 = regr_2.predict(X_test)
y_3 = regr_3.predict(X_test)

# Plot the results
plt.figure()
s = 50
plt.scatter(y[:, 0], y[:, 1], c="navy", s=s, label="data")
plt.scatter(y_1[:, 0], y_1[:, 1], c="cornflowerblue", s=s, label="max_depth=2")
plt.scatter(y_2[:, 0], y_2[:, 1], c="c", s=s, label="max_depth=5")
plt.scatter(y_3[:, 0], y_3[:, 1], c="orange", s=s, label="max_depth=8")
plt.xlim([-6, 6])
plt.ylim([-6, 6])
plt.xlabel("target 1")
plt.ylabel("target 2")
plt.title("Multi-output Decision Tree Regression")
plt.legend()
plt.show()
```

決策樹/範例三: Plot the decision surface of a decision tree on the iris dataset

http://scikit-learn.org/stable/auto_examples/tree/plot_iris.html#sphx-glr-auto-examples-tree-plot-iris-py

此範例利用決策樹分類器將資料集進行分類，找出各類別的分類邊界。以鳶尾花資料集當作範例，每次取兩個特徵做訓練，個別繪製不同品種的鳶尾花特徵的分布範圍。對於每對的鳶尾花特徵，決策樹學習推斷出簡單的分類規則，構成決策邊界。

範例目的：

1. 資料集：iris 鳶尾花資料集
2. 特徵：鳶尾花特徵
3. 預測目標：是哪一種鳶尾花
4. 機器學習方法：decision tree 決策樹

(一)引入函式庫及內建測試資料庫

- `from sklearn.datasets import load_iris` 將鳶尾花資料庫存入， `iris` 為一個dict型別資料。
- 每筆資料中有4個特徵，一次取2個特徵，共有6種排列方式。
- X (特徵資料) 以及 y (目標資料)。
- `DecisionTreeClassifier` 建立決策樹分類器。

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()

for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                [1, 2], [1, 3], [2, 3]]):
    X = iris.data[:, pair]
    y = iris.target
```

(二)建立Decision Tree分類器

建立模型及分類器訓練

- `DecisionTreeClassifier()` :決策樹分類器。
- `fit(特徵資料, 目標資料)` ：利用特徵資料及目標資料對分類器進行訓練。

```
clf = DecisionTreeClassifier().fit(X, y)
```

(三)繪製決策邊界及訓練點

- `np.meshgrid` ：利用特徵之最大最小值，建立預測用網格 xx, yy
- `clf.predict` ：預估分類結果。
- `plt.contourf` ：繪製決策邊界。
- `plt.scatter(X,y)` ：將X、y以點的方式繪製於平面上，c為數據點的顏色，label為圖例。

```
plt.subplot(2, 3, pairidx + 1)

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                     np.arange(y_min, y_max, plot_step))


Z = clf.predict(np.c_[xx.ravel(), yy.ravel()]) #np.c_ 串接兩個list,np.ravel將矩陣變爲一維

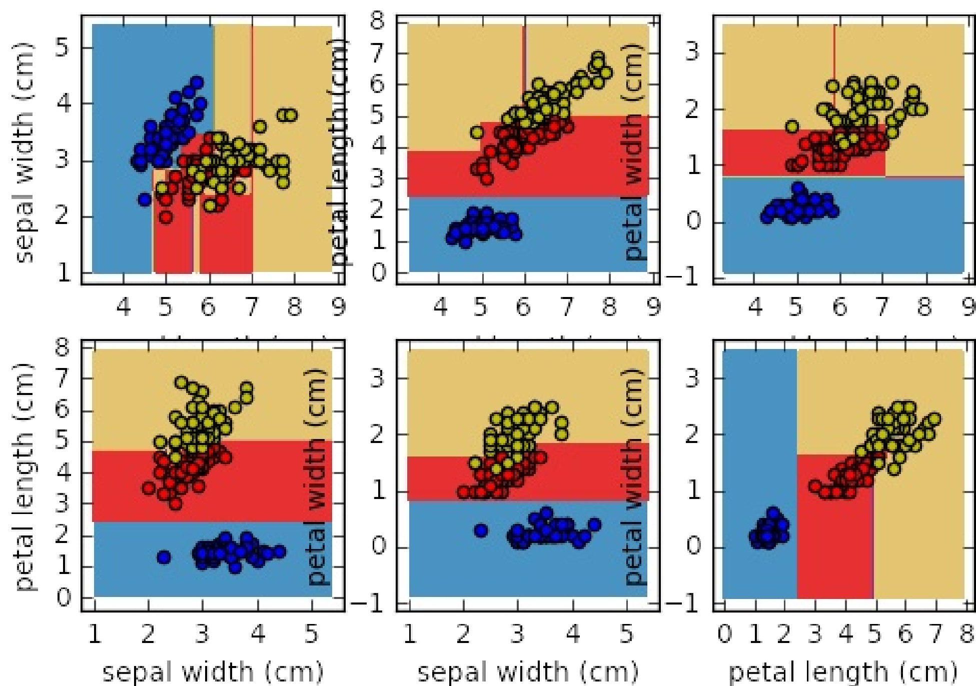
Z = Z.reshape(xx.shape)

cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)

plt.xlabel(iris.feature_names[pair[0]])
plt.ylabel(iris.feature_names[pair[1]])
plt.axis("tight")

for i, color in zip(range(n_classes), plot_colors):
    idx = np.where(y == i)
    plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
                cmap=plt.cm.Paired)

plt.axis("tight")
```



(四)完整程式碼

```
print(__doc__)
```

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

# Parameters
n_classes = 3
plot_colors = "bry"
plot_step = 0.02

# Load data
iris = load_iris()

for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                [1, 2], [1, 3], [2, 3]]):

    # We only take the two corresponding features
    X = iris.data[:, pair]
    y = iris.target
    # Train
    clf = DecisionTreeClassifier().fit(X, y)

    # Plot the decision boundary
    plt.subplot(2, 3, pairidx + 1)

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                         np.arange(y_min, y_max, plot_step))


    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()]) #np.c_ 串接兩個list,np.ravel將矩陣變爲一維

    Z = Z.reshape(xx.shape)


    cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)

    plt.xlabel(iris.feature_names[pair[0]])
    plt.ylabel(iris.feature_names[pair[1]])
    plt.axis("tight")


    # Plot the training points
    for i, color in zip(range(n_classes), plot_colors):
        idx = np.where(y == i)
        plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
                    cmap=plt.cm.Paired)

    plt.axis("tight")
```

```
plt.suptitle("Decision surface of a decision tree using paired features")
plt.legend()
plt.show()
```

決策樹範例四: Understanding the decision tree structure

http://scikit-learn.org/stable/auto_examples/tree/plot_unveil_tree_structure.html#sphx-glr-auto-examples-tree-plot-unveil-tree-structure-py

範例目的

此範例主要在進一步探討決策樹內部的結構，分析以獲得特徵與目標之間的關係，並進而進行預測。

1. 當每個節點的分支最多只有兩個稱之爲二元樹結構。
2. 判斷每個深度的節點是否爲葉，在二元樹中若該節點爲判斷的最後一層稱之爲葉。
3. 利用 `decision_path` 獲得決策路徑的資訊。
4. 利用 `apply` 得到預測結果，也就是決策樹最後抵達的葉。
5. 建立完成後的規則變能用來預測。
6. 一組多個樣本可以尋得其中共同的決策路徑。

(一)引入函式庫及測試資料

引入函式資料庫

- `load_iris` 引入鳶尾花資料庫。

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
```

建立訓練、測試集及決策樹分類器

- X (特徵資料) 以及 y (目標資料)。
- `train_test_split(X, y, random_state)` 將資料隨機分爲測試集及訓練集。
  X爲特徵資料集、y爲目標資料集， `random_state` 隨機數生成器。
- `DecisionTreeClassifier(max_leaf_nodes, random_state)` 建立決策樹分類器。
  `max_leaf_nodes` 節點爲葉的最大數目， `random_state` 若存在則爲隨機數生成器，若不存在則使用 `np.random` 。
- `fit(X, y)` 用做訓練，X爲訓練用特徵資料，y爲目標資料。

```
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
estimator = DecisionTreeClassifier(max_leaf_nodes=3, random_state=0)
estimator.fit(X_train, y_train)
```

(二) 決策樹結構探討

在 `DecisionTreeClassifier` 中有個屬性 `tree_` ，儲存了整個樹的結構。
二元樹被表示爲多個平行的矩陣，每個矩陣的第i個元素儲存著關於節點"i"的信息，節點0代表樹的根。
需要注意的是，有些矩陣只適用於有分支的節點，在這種情況下，其他類型的節點的值是任意的。

上述所說的矩陣包含了：

1. `node_count` ：總共的節點個數。

2. `children_left` ：節點左邊的節點的ID，"-1"代表該節點底下已無分支。
3. `children_righ` ：節點右邊的節點的ID，"-1"代表該節點底下已無分支。
4. `feature` ：使節點產生分支的特徵，"-2"代表該節點底下已無分支。
5. `threshold` ：節點的閾值。若距離不超過 threshold ，則邊的兩端就視作同一個群集。

```
n_nodes = estimator.tree_.node_count
children_left = estimator.tree_.children_left
children_right = estimator.tree_.children_right
feature = estimator.tree_.feature
threshold = estimator.tree_.threshold
```

以下爲各矩陣的內容

```
n_nodes = 5
children_left [ 1 -1  3 -1 -1]
children_right [ 2 -1  4 -1 -1]
feature [ 3 -2  2 -2 -2]
threshold [ 0.80000001 -2.          4.94999981 -2.         -2.        ]
```

二元樹的結構所通過的各個屬性是可以被計算的，例如每個節點的深度以及是否爲樹的最底層。

- `node_depth` ：節點在決策樹中的深度(層)。
- `is_leaves` ：該節點是否爲決策樹的最底層(葉)。
- `stack` ：存放尚未判斷是否達決策樹底層的節點資訊。

將stack的一組節點資訊pop出來，判斷該節點的左邊節點ID是否等於右邊節點ID。
若不相同分別將左右節點的資訊加入stack中，若相同則該節點已達底層 `is_leaves` 設爲True。

```
node_depth = np.zeros(shape=n_nodes)
is_leaves = np.zeros(shape=n_nodes, dtype=bool)

stack = [(0, -1)]  #initial

while len(stack) > 0:
    node_id, parent_depth = stack.pop()
    node_depth[node_id] = parent_depth + 1

    # If we have a test node
    if (children_left[node_id] != children_right[node_id]):
        stack.append((children_left[node_id], parent_depth + 1))
        stack.append((children_right[node_id], parent_depth + 1))
    else:
        is_leaves[node_id] = True
```

執行過程

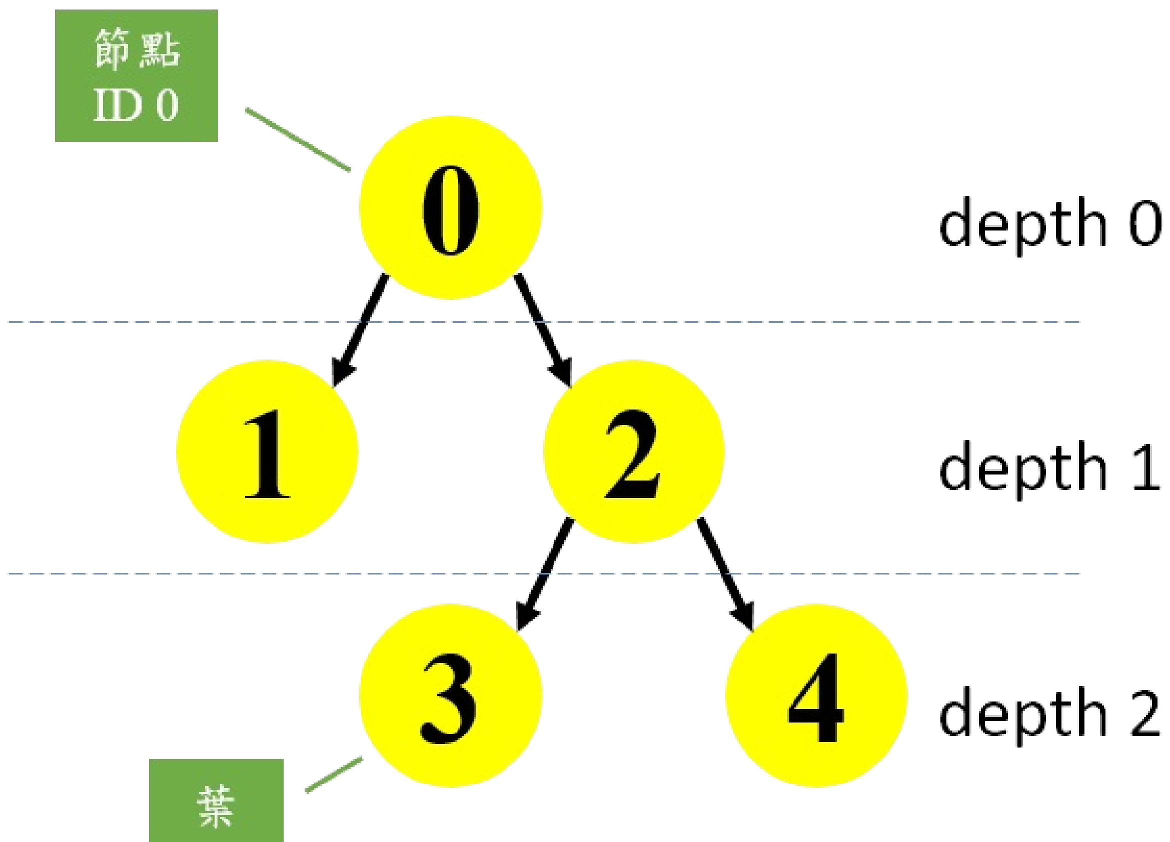
```
stack len 1
node_id 0 parent_depth -1
node_depth [ 0.  0.  0.  0.  0.]
stack [(1, 0), (2, 0)]

stack len 2
node_id 2 parent_depth 0
node_depth [ 0.  0.  1.  0.  0.]
stack [(1, 0), (3, 1), (4, 1)]

stack len 3
node_id 4 parent_depth 1
node_depth [ 0.  0.  1.  0.  2.]
stack [(1, 0), (3, 1)]

stack len 2
node_id 3 parent_depth 1
node_depth [ 0.  0.  1.  2.  2.]
stack [(1, 0)]

stack len 1
node_id 1 parent_depth 0
node_depth [ 0.  1.  1.  2.  2.]
stack []
```



下面這個部分是以程式的方式印出決策樹結構，這個決策樹共有5個節點。
若遇到的是test node則用閥值決定該往哪個節點前進，直到走到葉爲止。

```
print("The binary tree structure has %s nodes and has "
      "the following tree structure:"
      % n_nodes)
for i in range(n_nodes):
    if is_leaves[i]:
        print("%snode=%s leaf node." % (node_depth[i] * "\t", i)) #"\t"縮排
    else:
        print("%snode=%s test node: go to node %s if X[:, %s] <= %s else to "
              "node %s."
              % (node_depth[i] * "\t",
                 i,
                 children_left[i],
                 feature[i],
                 threshold[i],
                 children_right[i],
                 ))
```

執行結果

```
The binary tree structure has 5 nodes and has the following tree structure:
node=0 test node: go to node 1 if X[:, 3] <= 0.800000011921 else to node 2.
    node=1 leaf node.
    node=2 test node: go to node 3 if X[:, 2] <= 4.94999980927 else to node 4.
        node=3 leaf node.
        node=4 leaf node.
```

接下來要來探索每個樣本的決策路徑，利用 `decision_path` 方法可以讓我們得到這些資訊， `apply` 存放所有sample最後抵達哪個葉。

以第0筆樣本當作範例， `indices` 存放每個樣本經過的節點， `indptr` 存放每個樣本存放節點的位置， `node_index` 中存放了第0筆樣本所經過的節點ID。

```
node_indicator = estimator.decision_path(X_test)

# Similarly, we can also have the leaves ids reached by each sample.

leave_id = estimator.apply(X_test)

# Now, it's possible to get the tests that were used to predict a sample or
# a group of samples. First, let's make it for the sample.

sample_id = 0
node_index = node_indicator.indices[node_indicator.indptr[sample_id]:
                                    node_indicator.indptr[sample_id + 1]]
print('node_index', node_index)
print('Rules used to predict sample %s: ' % sample_id)
for node_id in node_index:
    if leave_id[sample_id] != node_id:
        continue

    if (X_test[sample_id, feature[node_id]] <= threshold[node_id]):
        threshold_sign = "<="
    else:
        threshold_sign = ">"

    print("decision id node %s : (X[%s, %s] (= %s) %s %s)"
          % (node_id,
             sample_id,
             feature[node_id],
             X_test[i, feature[node_id]],
             threshold_sign,
             threshold[node_id]))
```

執行結果

```
node_index [0 2 4]
Rules used to predict sample 0:
decision id node 4 : (X[0, -2] (= 1.5) > -2.0)
```

接下來是探討多個樣本，是否有經過相同的節點。
以樣本0、1當作範例， `node_indicator.toarray()` 存放多個矩陣0代表沒有經過該節點，1代表經過該節點。 `common_nodes` 中存放true與false，若同一個節點相加的值等於輸入樣本的各樹，則代表該節點都有被經過。

```
# For a group of samples, we have the following common node.
sample_ids = [0, 1]
common_nodes = (node_indicator.toarray()[sample_ids].sum(axis=0) ==
                len(sample_ids))

print('node_indicator',node_indicator.toarray()[sample_ids])
print('common_nodes',common_nodes)

common_node_id = np.arange(n_nodes)[common_nodes]
print('common_node_id',common_node_id)


print("\nThe following samples %s share the node %s in the tree"
      % (sample_ids, common_node_id))
print("It is %s %% of all nodes." % (100 * len(common_node_id) / n_nodes,))
```

執行結果

```
node_indicator [[1 0 1 0 1]
 [1 0 1 1 0]]
common_nodes [ True False  True False False]
common_node_id [0 2]

The following samples [0, 1] share the node [0 2] in the tree
It is 40.0 % of all nodes.
```

(三)完整程式碼

```
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

estimator = DecisionTreeClassifier(max_leaf_nodes=3, random_state=0)
estimator.fit(X_train, y_train)

# The decision estimator has an attribute called tree_  which stores the entire
# tree structure and allows access to low level attributes. The binary tree
# tree_ is represented as a number of parallel arrays. The i-th element of each
# array holds information about the node `i`. Node 0 is the tree's root. NOTE:
# Some of the arrays only apply to either leaves or split nodes, resp. In this
# case the values of nodes of the other type are arbitrary!
#
# Among those arrays, we have:
#   - left_child, id of the left child of the node
```

```
#    - right_child, id of the right child of the node
#    - feature, feature used for splitting the node
#    - threshold, threshold value at the node
#

# Using those arrays, we can parse the tree structure:

n_nodes = estimator.tree_.node_count
children_left = estimator.tree_.children_left
children_right = estimator.tree_.children_right
feature = estimator.tree_.feature
threshold = estimator.tree_.threshold


# The tree structure can be traversed to compute various properties such
# as the depth of each node and whether or not it is a leaf.
node_depth = np.zeros(shape=n_nodes)
is_leaves = np.zeros(shape=n_nodes, dtype=bool)
stack = [(0, -1)]  # seed is the root node id and its parent depth
while len(stack) > 0:
    node_id, parent_depth = stack.pop()
    node_depth[node_id] = parent_depth + 1

    # If we have a test node
    if (children_left[node_id] != children_right[node_id]):
        stack.append((children_left[node_id], parent_depth + 1))
        stack.append((children_right[node_id], parent_depth + 1))
    else:
        is_leaves[node_id] = True

print("The binary tree structure has %s nodes and has "
      "the following tree structure:"
      % n_nodes)
for i in range(n_nodes):
    if is_leaves[i]:
        print("%snode=%s leaf node." % (node_depth[i] * "\t", i))
    else:
        print("%snode=%s test node: go to node %s if X[:, %s] <= %ss else to "
              "node %s."
              % (node_depth[i] * "\t",
                 i,
                 children_left[i],
                 feature[i],
                 threshold[i],
                 children_right[i],
                 ))
print()

# First let's retrieve the decision path of each sample. The decision_path
# method allows to retrieve the node indicator functions. A non zero element of
# indicator matrix at the position (i, j) indicates that the sample i goes
# through the node j.
```

```
node_indicator = estimator.decision_path(X_test)

# Similarly, we can also have the leaves ids reached by each sample.

leave_id = estimator.apply(X_test)

# Now, it's possible to get the tests that were used to predict a sample or
# a group of samples. First, let's make it for the sample.

sample_id = 0
node_index = node_indicator.indices[node_indicator.indptr[sample_id]:
                                    node_indicator.indptr[sample_id + 1]]

print('Rules used to predict sample %s: ' % sample_id)
for node_id in node_index:
    if leave_id[sample_id] != node_id:
        continue

    if (X_test[sample_id, feature[node_id]] <= threshold[node_id]):
        threshold_sign = "<="
    else:
        threshold_sign = ">"

    print("decision id node %s : (X[%s, %s] (= %s) %s %s)"
          % (node_id,
             sample_id,
             feature[node_id],
             X_test[i, feature[node_id]],
             threshold_sign,
             threshold[node_id]))

# For a group of samples, we have the following common node.
sample_ids = [0, 1]
common_nodes = (node_indicator.toarray()[sample_ids].sum(axis=0) ==
                len(sample_ids))

common_node_id = np.arange(n_nodes)[common_nodes]

print("\nThe following samples %s share the node %s in the tree"
      % (sample_ids, common_node_id))
print("It is %s %% of all nodes." % (100 * len(common_node_id) / n_nodes,))
```

機器學習：使用 **NVIDIA Jetson TX2**

從零開始

灌作業系統一定是我們的首要目標，但在這之前，我們要先有一台運行 Ubuntu x64 (14.04或更新) 的電腦，可以用虛擬機來代替。

沒有虛擬機的朋友可以用VirtualBox。

Ubuntu x64 的映像檔可以在這邊下載。

1. 安裝 VirtualBox

流程就不在這邊贅述，簡單來說，就是狂按下一步。

2. 安裝 Ubuntu x64

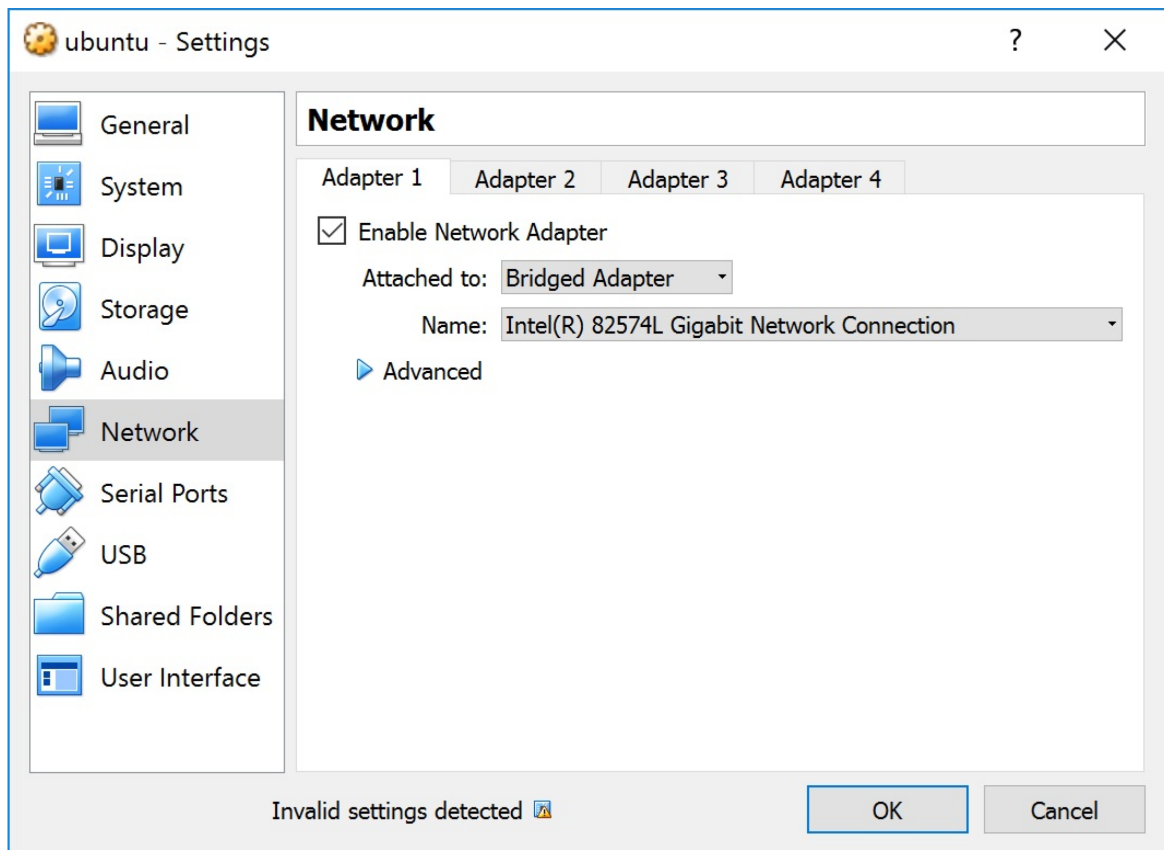
有兩點要注意：

1. 因爲稍後下載回來的安裝包還會有額外的套件需要下載與編譯，所以硬碟空間要大一些，建議設定在 **50G** 以上。

2. 在設定 TX2 的時候，需要將 TX2 連上與 Ubuntu x64 相同 的區網，進行這項設定，請：

    i. 開啓 Ubuntu x64 的虛擬機設定
    ii. 進入網路設定

iii. 將網路連線方式選擇橋接，並指定主機用來上網的網路介面



讓 TX2 動起來

基本上外部的設置已經完成了，接下來就要把目光轉移到 TX2 上面。

這邊我們會用到名爲 Jet Pack 的官方套件，可以在這邊下載他。

1. 執行 JetPack

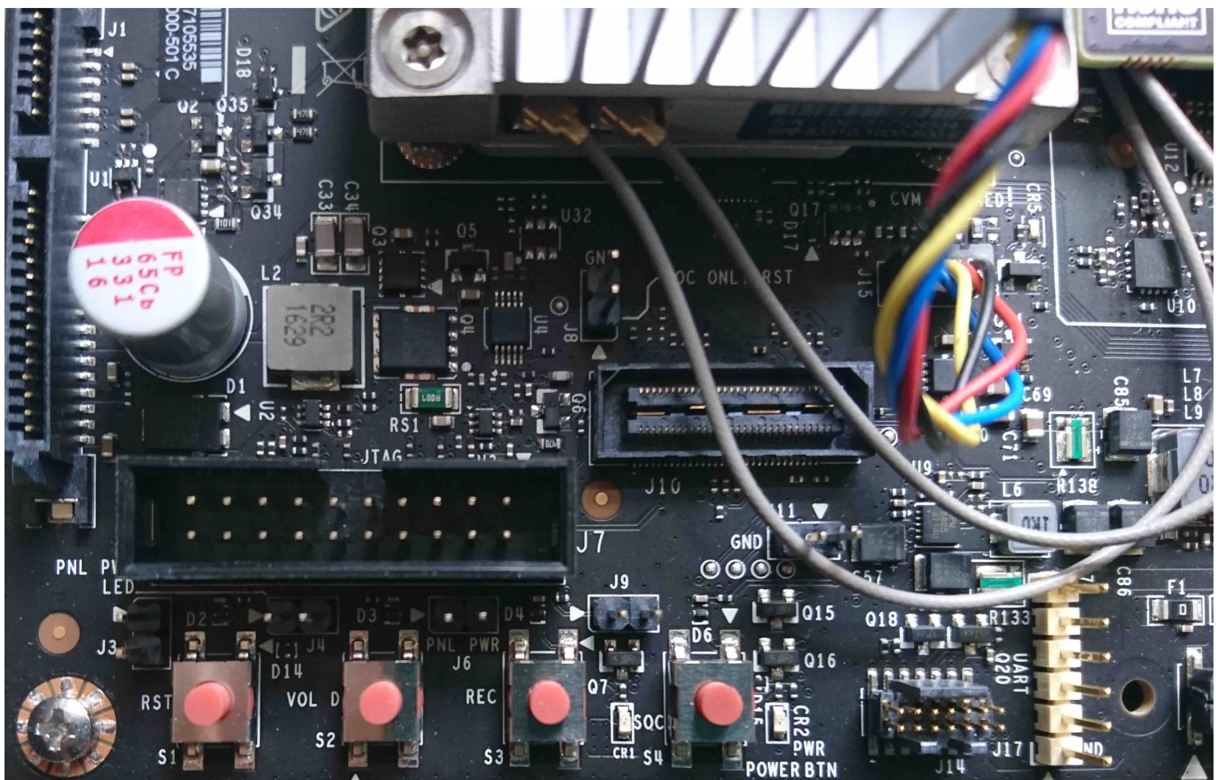
注意：這個套件要在 **Ubuntu x64** 上才能執行

首先，我們需要更改 JetPack 的權限，讓他可以執行：

1. 開啓 JetPack 所在的資料夾。
2. 點右鍵，選 `Open in Terminal` 。
3. 執行 `chmod +x JetPack-<VERSION>.run` ，其中的 `<VERSION>` 請替換成你所下載的版本號。（可以按 `tab` 讓系統自動補齊檔案名稱）

基本上接下來按照指示操作即可，當出現 Terminal 視窗時：

1. 除了要將 TX2 準備在旁邊之外，你還會需要：可以和隨附的變壓器搭配的電源線、隨附的**micro-USB**線、夠長且良好的網路線、可以用**HDMI**或有HDMI轉接線可以搭配的螢幕、鍵盤、滑鼠。
2. 將 TX2 接上電源、用**micro-USB**線把 TX2 和 Ubuntu x64 連在一起、網路線接到和 Ubuntu x64 相同的區域網路下。
3. 在按按鈕之前，先來介紹按鈕的作用

   在電源接頭對角處有**1**個螺絲和**4**顆按鈕，他們分別是

   |螺絲|重設|自訂|復原|電源| |---|---|---|---|---| | | | | | |

4. 按下電源開機
5. 按住復原別放開
6. 按下重設進入復原模式
7. 可以放開復原鍵
8. 確認 Ubuntu x64 有成功辨識 TX2，名稱會有 **NVIDIA** 或 **Jetson** 或 **TX2** 字樣

9. 在 Terminal 內按下 `enter` 鍵，繼續程序
10. 等待完成的訊息出現

2. 安裝環境

先來列出預設的帳號與密碼：

帳號	密碼
nvidia	nvidia
ubuntu	ubuntu

接下來的事情，我們都會在 Terminal 裡面完成：

1. 先更新系統

```
$ sudo apt-get update
$ sudo apt-get upgrade -y
```

2. 接著安裝常用的套件

```
$ sudo apt-get install curl vim git mercurial silversearcher-ag htop python3-pip
$ pip3 install --upgrade pip
$ pip3 install virtualenv numpy
```

3. (Optional)安裝更方便的 zsh

```
$ sudo apt-get install zsh
$ sh -c "$(curl -fsSL https://raw.githubusercontent.com/robbyrussell/oh-my-zsh/master/tools/install.sh)"
```

安裝 OpenCV

既然 TX2 上面有相機模組，那我們就來裝個 OpenCV 來做相機的影像處理吧！

Python3 會是我們的主要語言。

1. 安裝依賴套件

```
$ sudo apt-get install build-essential cmake git pkg-config libjpeg8-dev libtiff5-dev libjasper-dev libpng12-dev libavcodec-dev libavformat-dev libswscale-dev libv4l-dev libgtk2.0-dev libatlas-base-dev gfortran
```

2. 取得 OpenCV 原始碼

```
$ git clone git://github.com/Itseez/opencv
$ cd opencv
$ git checkout <你所要用的 OpenCV 版本，建議是用最新版>
$ git clone git://github.com/Itseez/opencv_contrib
$ cd opencv_contrib
$ git checkout <你所要用的 OpenCV 版本，建議是用最新版>
```

3. 編譯 OpenCV

```
$ mkdir opencv/build
$ cd opencv/build
$ cmake \
  -D CMAKE_BUILD_TYPE=RELEASE \
  -D CMAKE_INSTALL_PREFIX=/usr/local \
  -D INSTALL_PYTHON_EXAMPLES=ON \
  -D OPENCV_EXTRA_MODULES_PATH=<opencv_contrib 所在的完整路徑> \
  -D BUILD_EXAMPLES=ON ..
$ make -j4
$ sudo make install
$ sudo ldconfig
```

4. 安裝 OpenCV

   i. 建立虛擬開發環境

   ```
   $ virtualenv OpenCV
   ```

   ii. 進入虛擬開發環境

   ```
   $ cd OpenCV
   $ source bin/active
   ```

   iii. 找到 OpenCV

   ```
   $ ls -al /usr/local/lib/python<Python3 的版本號，如：3.5>/dist-packages/cv2*
   ```

   iv. 連接 OpenCV 到虛擬環境

```
$ ln -s <上一步驟印出的完整路徑> lib/python<Python3 的版本號，如：3.5>/site-packages/cv2.so
```

v. 測試 OpenCV

```
import cv2
cv2.__version__
```

輸出

```
'<OpenCV 的版本號>'
```

安裝 TensorFlow

1. 安裝依賴套件

```
$ sudo apt-get install default-jdk libcupti-dev
$ export JAVA_HOME='/usr/lib/jvm/java-8-openjdk-arm64/'
```

2. 取得 TensorFlow 編譯腳本

```
$ git clone git://github.com/jetsonhacks/installTensorFlowTX2
$ cd installTensorFlowTX2
```

3. 執行編譯腳本

```
$ ./installPrerequisitesPy3.sh
$ ./cloneTensorFlow.sh
$ ./setTensorFlowEVPy3.sh
$ ./buildTensorFlow.sh
$ ./packageTensorFlow.sh
```

4. 安裝 TensorFlow
   i. 建立虛擬開發環境

   ```
   $ virtualenv TensorFlow
   ```

   ii. 進入虛擬開發環境

   ```
   $ cd TensorFlow
   $ source bin/active
   ```

   iii. 安裝 TensorFlow 到虛擬環境

   ```
   pip3 install $HOME/<TensorFlow 的 .whl 安裝封包>
   ```

   iv. 測試 TensorFlow
      i. Hello World

      ```
      import tensorflow as tf
      hello = tf.constant('Hello, TensorFlow on NVIDIA Jetson TX2!')
      sess = tf.Session()
      print(sess.run(hello))
      ```

      輸出

      ```
      Hello, TensorFlow on NVIDIA Jetson TX2!
      ```

      ii. 運算單元

```
import tensorflow as tf
# Creates a graph.
a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
c = tf.matmul(a, b)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(c))
```

輸出

```
name: NVIDIA Tegra X2
major: 6 minor: 2 memoryClockRate (GHz) 1.3005
MatMul: (MatMul): /job:localhost/replica:0/task:0/gpu:0
b: (Const): /job:localhost/replica:0/task:0/gpu:0
a: (Const): /job:localhost/replica:0/task:0/gpu:0
[[ 22.  28.]
[ 49.  64.]]
```